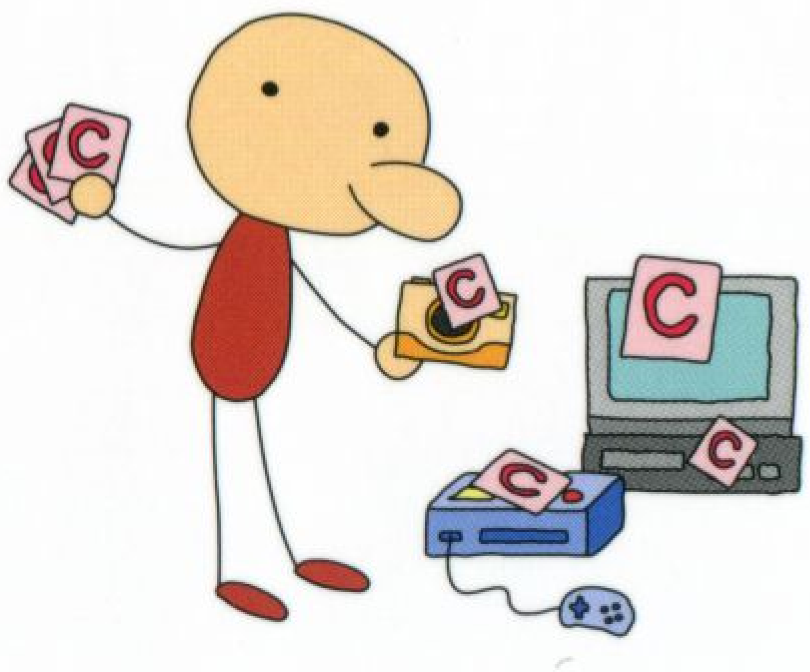
The basic guide book of C Programming for beginners

世界でいちばん簡単な

C言語のe(イー)本

C言語の基本と考え方がわかる本

Sachio Horie
堀江幸生＝著
with Anchor Pro



- ★オールカラー、多彩なイラストでC言語が直感的にわかる!
- ★丁寧な解説でプログラムの仕組みが誰でもわかる!
- ★豊富な例題と練習問題でC言語の基礎固めができる!
- ★アルゴリズム、関数とポインタの考え方がわかる!
- ★四コマ漫画やQ&Aなど、楽しくプログラムが学べる!
- ★じゃんけんゲームサンプルプログラムダウンロードサービス付き!

秀和システム

世界でいちばん簡単な C言語のe本

イー

C言語の基本と考え方がわかる本

Sachio Horie
堀江幸生＝著
with Anchor Pro



●注意

❶本書は著者が独自に調査した結果を出版したものです。

❷本書は内容について万全を期して作成していたしましたが、万一、ご不審な点や誤り、記載漏れなど、お気付きの点がありましたら、出版元まで書面にてご連絡ください。

❸本書の内容に関して運用した結果の影響については、上記❷項にかかわらず責任を負いかねます。あらかじめご了承ください。

❹本書の全部、または一部について、出版元から文書による許諾を得ずに複製することは禁じられています。

❺商標

・Microsoft. Windowsは、米国Microsoft Corporationおよび、その他の国における商標、または登録商標です。

・その他、ソフトウェア名、ハードウェア名は一般に各メーカーの商標、または登録商標です。

はじめに

あなたは、C言語をどうして学んでみようと思いましたか？

学校で習うからとか、仕事で必要になりそうだからとか、様々な理由があるでしょう。ここで大切なことは、あなたが、これからコンピュータと深くコミュニケートする機会を持ったということだと思います。

プログラミングすることは、ワープロで文書を作ったり、インターネットでチャットすることとは違って、コンピュータと対話することです。コンピュータは融通が利きません。もし、あなたが間違った指示を与えれば、期待どおりには動いてくれませんし、あなたの感情も無視されます。でも、これは言い換えると、あなたがコンピュータに正確な指示を与えれば、嫌とも言わず必ず期待どおりに動いてくれる、何とも健気な存在でもあるのです。人と対話することと違って、こんなコンピュータだから、最初は戸惑うことでしょうし、途中で投げ出したくなるかもしれませんが、そこはちょっと我慢してコンピュータと付き合ってみてください。必ず、コンピュータと対話することが楽しいと感じるときがきます。そのときが、本当にコンピュータと対話できたときです。

コンピュータにプログラミングする言語には、たくさんの種類がありますが、C言語は、たとえればブリティッシュ・イングリッシュみたいなものです（Javaは、アメリカン・イングリッシュかな）。歴史があって、使いこなすには難しいところもあるからです。ただし、本書では、プログラミングの考え方や他の言語でも共通な部分に焦点を当てて、C言語の難しいところは、さらっと流しています。それは、プログラミングの本として適切ではないという意見もあると思いますが、まずは、コンピュータとコミュニケートする楽しさを知ってもらうことが大切だと考えたからです。

本書は、はじめてコンピュータと深くコミュニケートする機会を持った人に、「プログラミングっておもしろいな」と言ってもらうこと、そのために書かれています。

付録の「じゃんけんゲームプログラム」のサンプルコードは、こちらのURLでダウンロードすることができますのでご利用ください。

URL:http://www.shuwasystem.co.jp/books/7980/0542-8/0542-8.html

2003年6月　堀江幸生

本書の特徴

▶対象読者

本書は、以下のような方を対象読者としています。

・中学生以上の方
・コンピュータに何ができて何ができないかを知りたい方
・仕事上、プログラミングを覚えなければならない方
・算数、数学は苦手だけれど、プログラミングに興味のある方
・一度プログラミングを挑戦したけれど、挫折してしまった方

▶本書の目的

本書の目的は、たった1冊の本でプログラミングという考え方のエッセンスを提供することにあります。

▶本書の構成

★各章の扉

本書は、C言語プログラムの理解を助けるために、各章の扉に読者を導くための分岐点（簡単な説明文）を設けています。本書を進めていく上でぜひ参考にしてください。

★第1章

この章では、プログラミング言語とはそもそも何なのかについて説明することからはじまって、C言語というプログラミング言語について簡単に紹介しています。

★第2章

この章では、C言語を使ってプログラミングするために、必要な環境や手順について説明しています。

★第3章

この章では、プログラミングという考え方についてやさしく説明しています。

★第4章

この章では、プログラミングに必要な知識を順に追って説明しています。ここでは、変数の使い方について説明しています。

★第5章

この章では、どうすればうまくコンピュータに指示を出せるのかについて、制御構造というものを使って考えます。

★第6章

この章では、C言語の中でも重要な関数とポインタについて紹介しています。

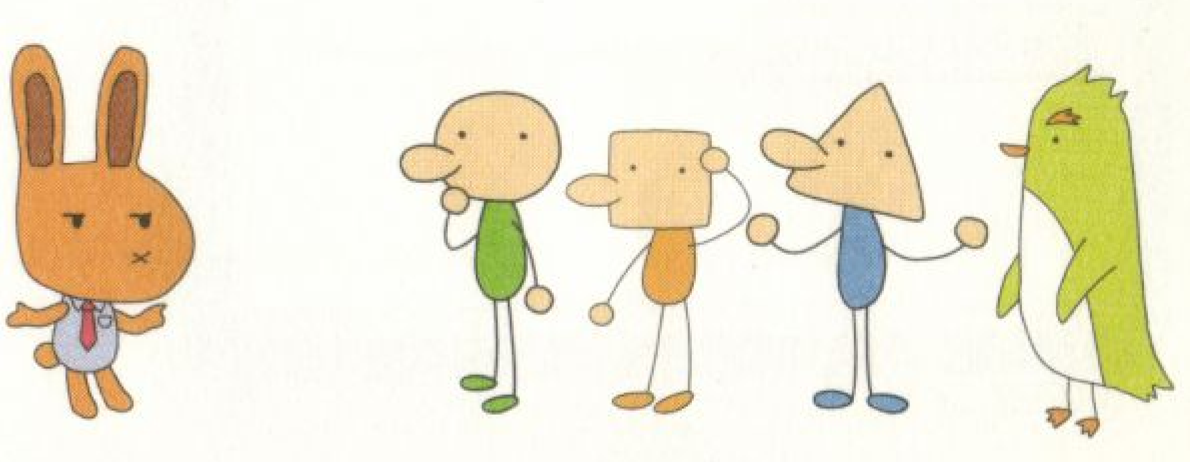
★第7章

この章では、よりプログラムらしいプログラムを作るために、これだけは知っておきたいというものを紹介しています。

推奨するプログラミング環境

本書は、次の環境でプログラミングしています。

・オペレーティングシステム：Windows XP
・gcc：Mingw gcc

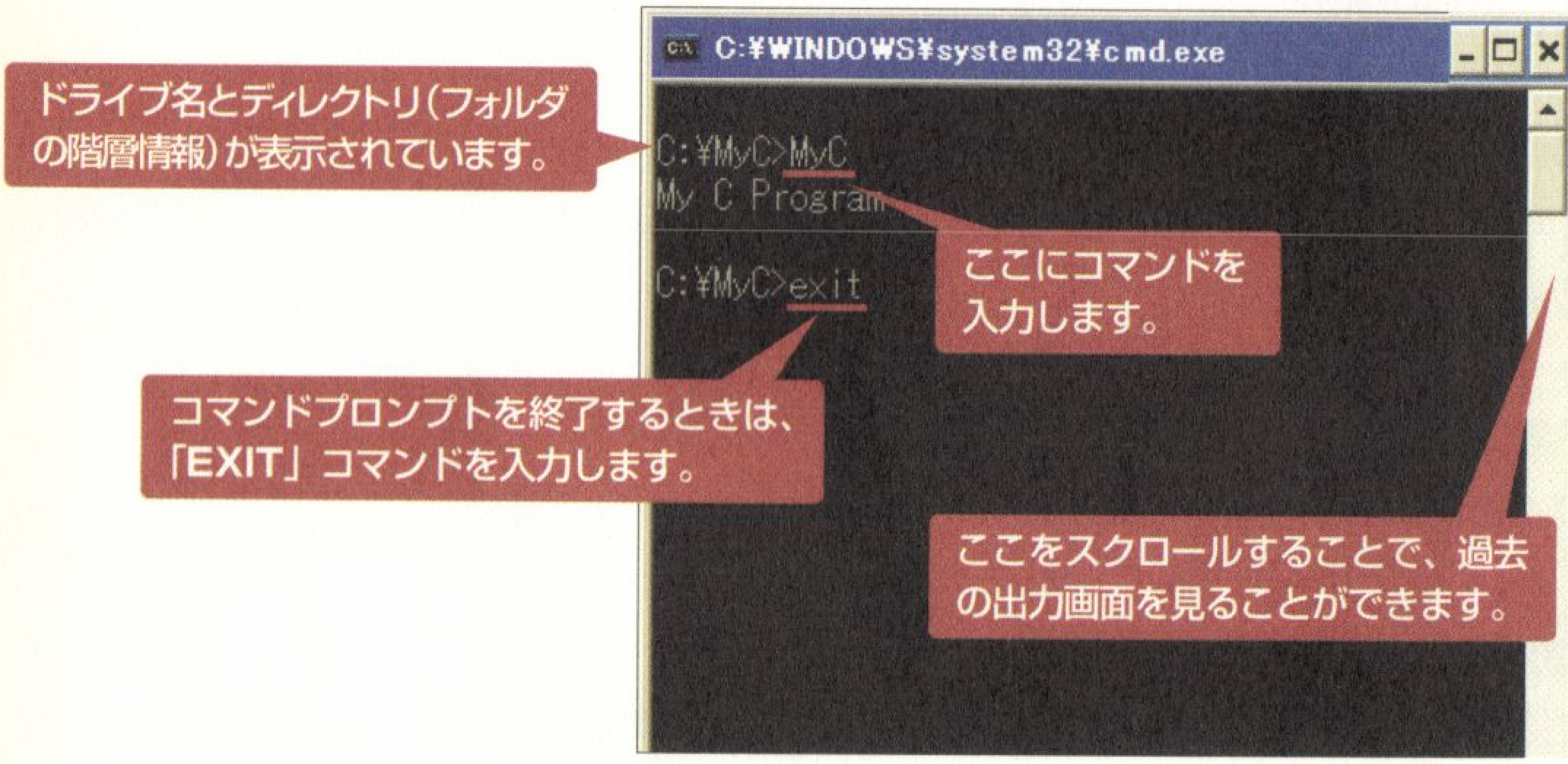


コマンドプロンプトの画面

C言語のコンパイルや実行は、コマンドプロンプトから行います。

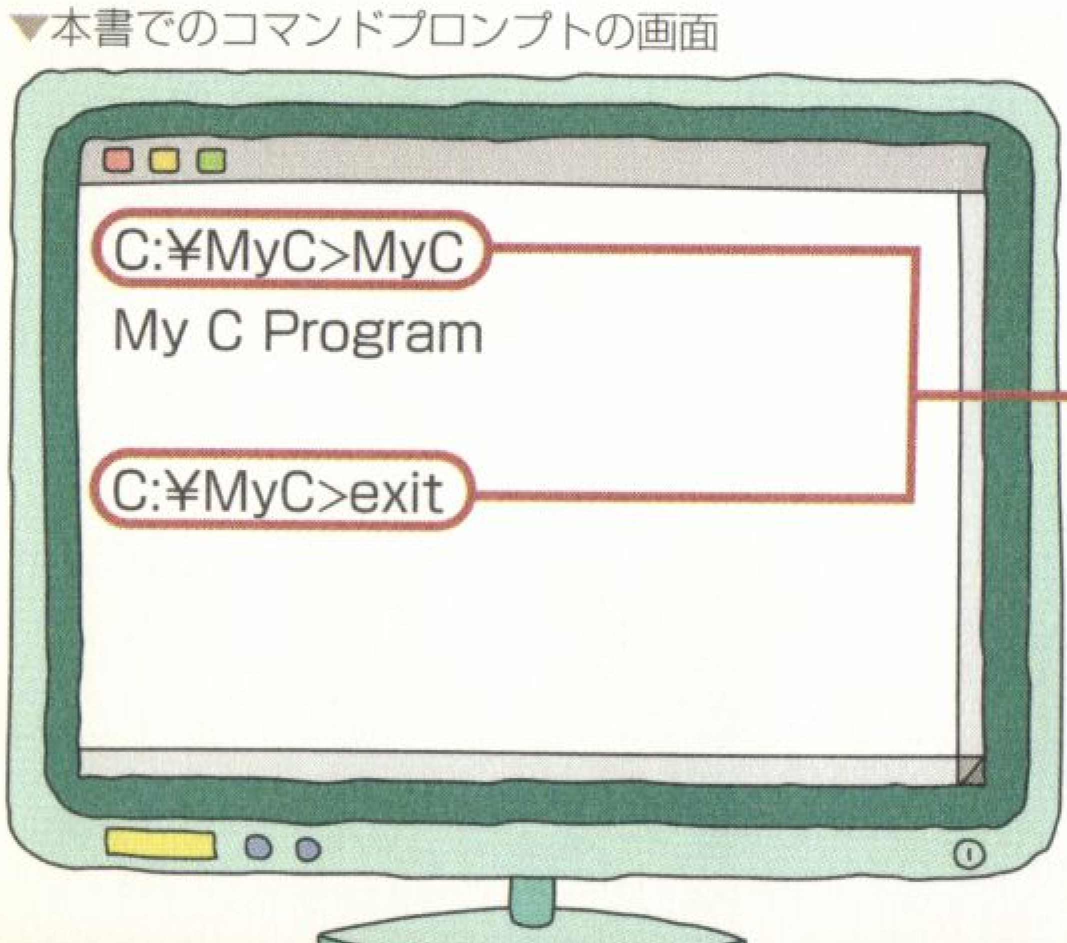
コマンドプロンプトの起動方法は、**スタート**メニューから、**プログラム➡アクセサリ➡コマンドプロンプト**（MS-DOSプロンプト）を選択するか、または**スタート**メニューから、**ファイル名を指定して実行➡**「**cmd**」と入力します。

画面では、プログラムがCドライブの「**MyC**」フォルダに保存してあるものとして実行しています。



本書ではコマンドプロンプトの画面を、次のようなイラストを使って表しています。
また、出力結果には、次のような記号を用いています。

▼本書でのコマンドプロンプトの画面



▼本書でのエラー画面

エラー

```
MyC.c: In function`main' :
MyC.c:6: parse error before`return'
```

目次

CONTENTS

導入編❶

第1章　プログラミング言語C言語について

導入編❷

第2章　プログラミングをはじめる前に

基本編

第3章　プログラミングの基本

応用編❶

第4章 プログラミングの心得❶

応用編❷

第5章 プログラミングの心得❷

実践編❶

第6章　関数とポインタ

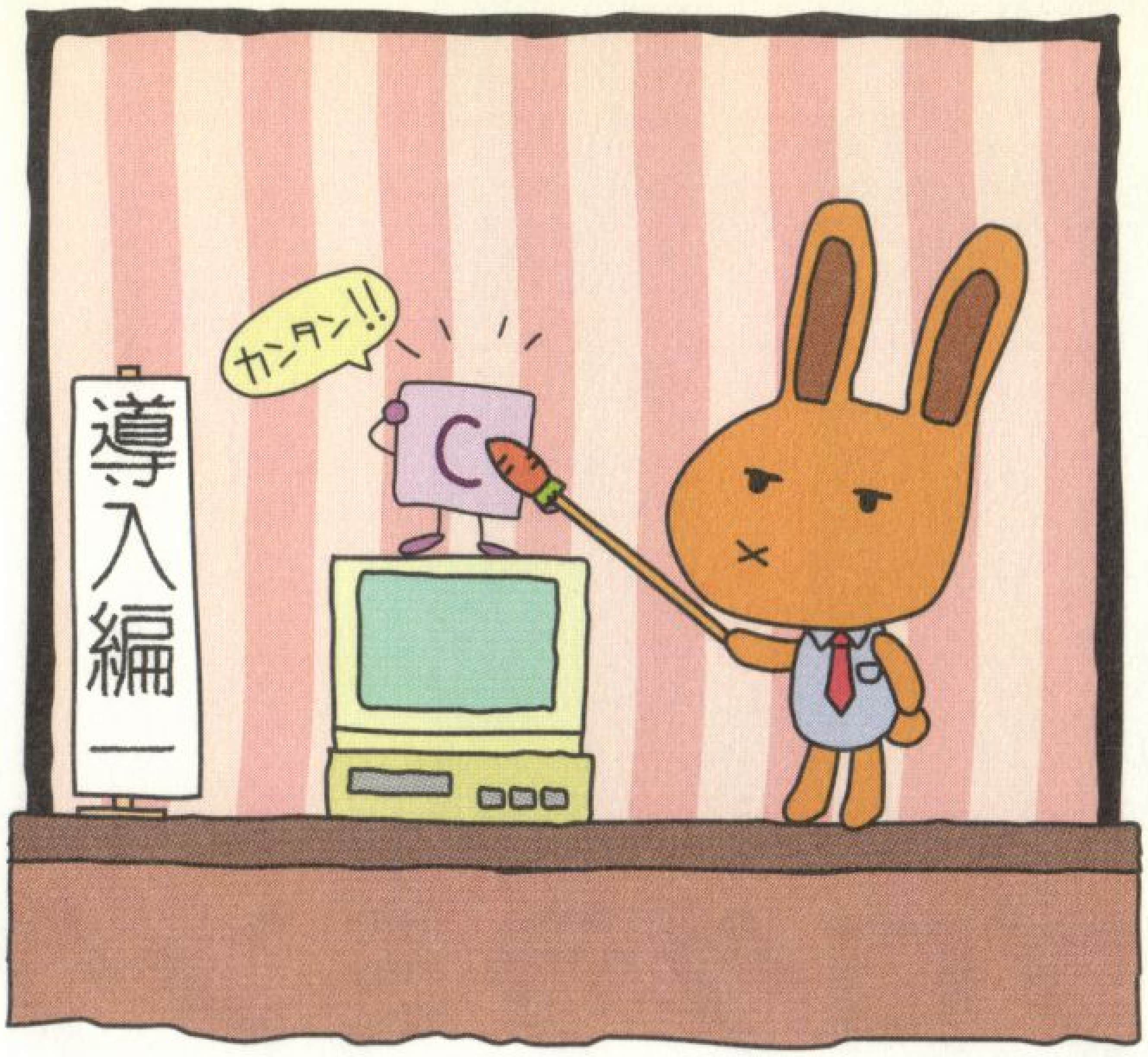
実践編❷

第7章　C言語を使う

付録

索引

コラム



★ 本書は、はじめてC言語プログラミングを学ぶ人や、
何度かプログラミングに挑戦したもののあきらめて
しまった人におすすめです。
「プログラミングは難しい」と思っている人も、
ぜひ挑戦してみてください。

★では、さっそくページを開いてみましょう。

登場キャラクターとともに、新しいC言語の世界を
体験することができるでしょう。

★すでにC言語プログラミングの準備ができていて、
C言語の基本から学びたい方は

➡ 第3章本文54ページへ

Chapter 1

プログラミング言語 C言語について



この章でマスターすること

プログラミング言語とは、人とコンピュータがコミュニケーションをとるために作られた人工の言語のことで、**C言語**もその中の一つです。

つまり、「C言語でプログラミングする」ということは、「C言語でコンピュータとコミュニケーションをとる」ということになるのです。

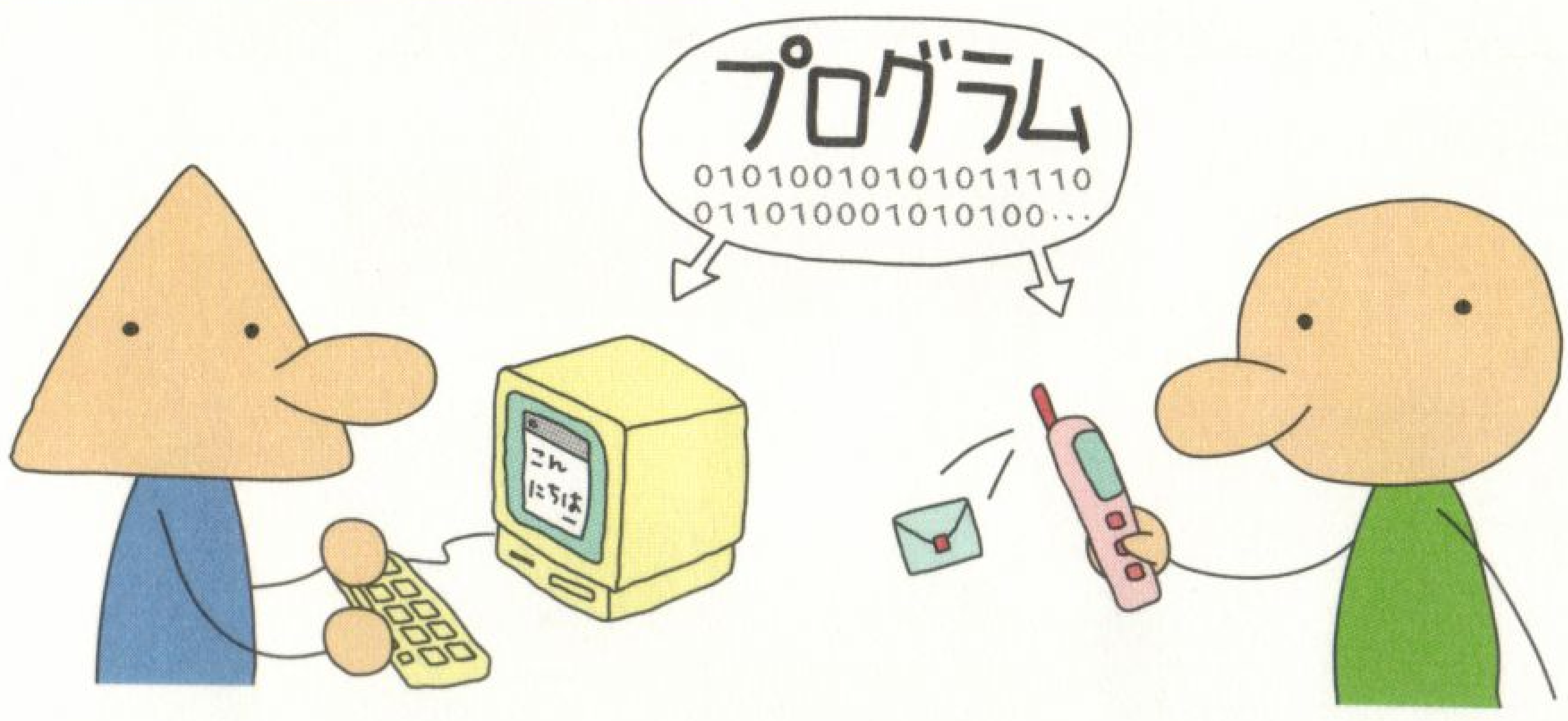
人とコンピュータとのコミュニケーションの結果が、**プログラム**です。これは、C言語というプログラミング言語の文の集まりで、コンピュータへの指示を書いたものです。

例えば、パソコンのワープロソフトで文章を入力したり、携帯電話でメールができるのは、実はコンピュータが理解できるようにかかれたプログラムのおかげなのです。



この章では、プログラムとC言語について学んでいきます。

プログラムって、いったい何だろう？



C言語必須用語

プログラミング言語
プログラム
入力と演算と出力
アプリケーション

ステップアップ用語

インタプリタ　**コンパイラ**
Unicode

Section 01 C言語とは何だろう

「C言語を使って、プログラミングする」とは、本当はどういうことなのでしょうか。ここでは、人とコンピュータとの関わりを通して、C言語で作るプログラムについて説明していきます。

プログラミング言語とは

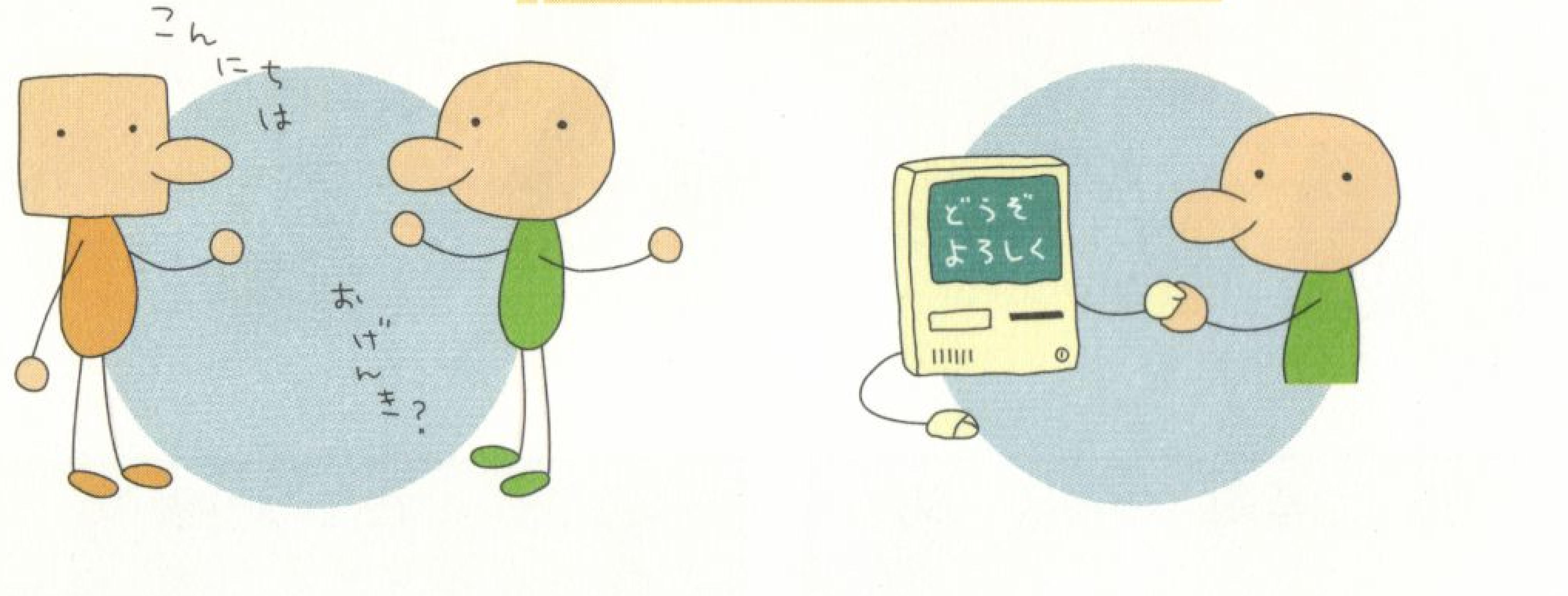
プログラミング言語とは、プログラムを開発するために用意された人工の言語のことです。もっとわかりやすくいうと、プログラミング言語とは、人とコンピュータがコミュニケーションするために作られた言語ともいえます。

プログラミング言語の**プログラミング**とは、コンピュータに「プログラムする」ということです。コンピュータに「プログラムする」とは、コンピュータにこれからしてもらいたいこと（作業）を指示する（プログラムを開発する）ことです。

例えば、人が誰かに作業を指示するときは、コミュニケーションをとる必要がありますが、その手段の一つとして言語があります。相手が日本人ならば、日本語を使いますし、そうでなければ英語やイタリア語など、その人に通じる他の言語を使ったりします。

同じように、人とコンピュータがコミュニケーションをとるときも、その手段の一つとして言語があるのです。

プログラム言語によってコンピュータとコミュニケーションをとる



C言語とは

C言語とは、数あるプログラミング言語の一つです。

人が使う言葉（**自然言語**といいます）が、考え方や国の違いによって日本語や英語など様々なものがあるように、プログラミング言語も様々なものが用意されています。これは、

コンピュータに指示する作業の内容やコンピュータの種類の違いによります。

例えば、給料をパソコンで計算すること、気象の予測をスーパーコンピュータですること、冷蔵庫やビデオなどの家電をマイクロコンピュータで制御することでは、コンピュータに指示する作業の内容も、そのことを扱うコンピュータの種類も大きく違ってきます。

そこで、場合に応じて、それぞれを処理するのに得意な言語を用意していったのです。

▼よく知られているプログラミング言語の種類

言語	用途	特徴
C	組み込みシステム※、ゲーム	ハードウェアよりの処理が得意です。
C++	ワープロなどのアプリケーション	C言語+オブジェクト指向です。
アセンブラ	組み込みシステム	ハードウェアを直接に制御できます。
Java	組み込みシステム、ネットワーク	機種を選びません。

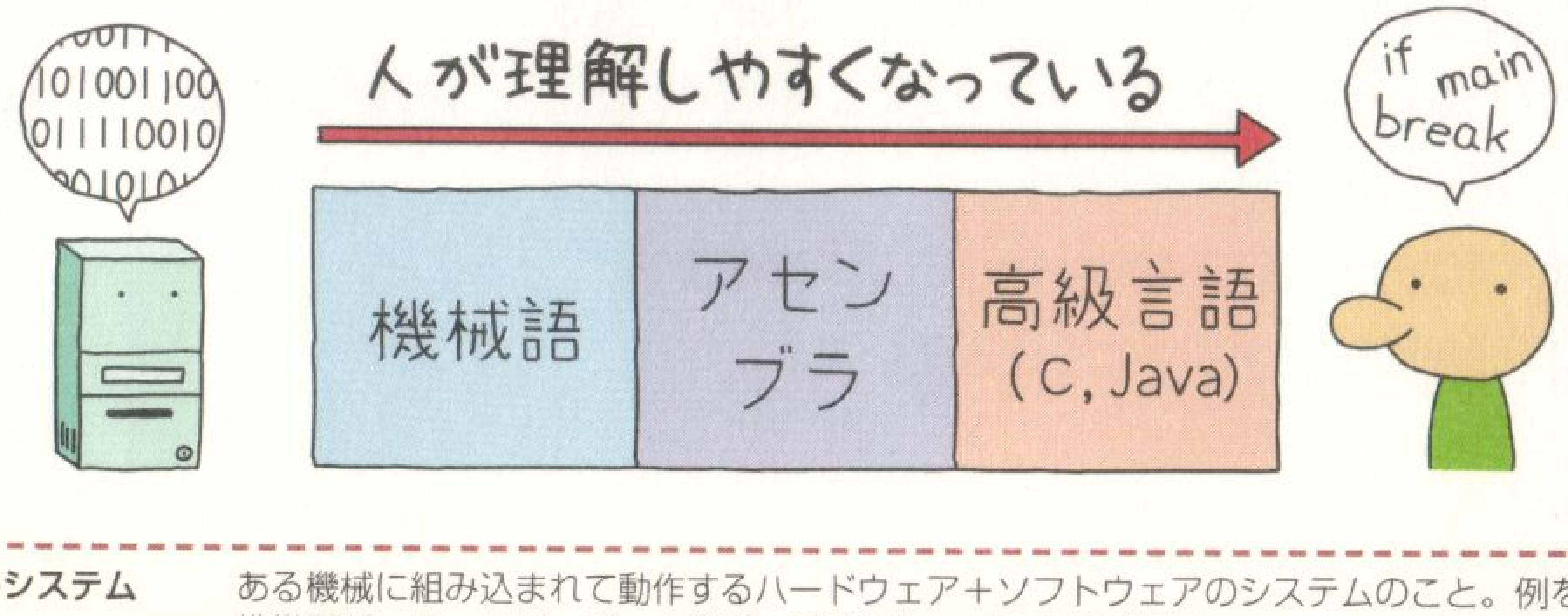
本書では、プログラミングを学習する上で、C言語は道具であり、目的ではありません。

▶専門的な説明

プログラミング言語の種類は、大きく分けて次の3つがあります。

▼プログラミング言語の種類

言語	解説
機械語	コンピュータが、直接解釈できる言語。0と1の並びで表現されているので、そのままでは人間には理解できません。
アセンブラ言語	機械語に対して、人間にわかるような命令を一対一に割り当てて作られた言語。機械語よりは、わかりやすく、直接ハードウェアを操作することができる利点があります。しかし、コンピュータのハードウェアに関する高度な知識が必要になります。
高級言語	アセンブラ言語より、さらに人間にわかりやすく設計された言語。最終的には、機械語に翻訳されます。



※組み込みシステム　ある機械に組み込まれて動作するハードウェア+ソフトウェアのシステムのこと。例をあげると、携帯電話、カーナビ、ゲーム機器、デジタルカメラなどがある。

C言語の誕生

C言語は、1972年にアメリカのAT&Tベル研究所で、Dennis M .Ritchie（リッチー）によって設計されました。彼の目的は、オペレーティングシステム（OS）のUNIXを開発することでしたが、Brian W.Kernighan（カーニハン）と共に、UNIXの開発にふさわしい言語としてC言語がどうあるべきかをまとめたのです。

当初、C言語はOS開発用でしたが、UNIXをはじめとする各種OS上で動くプログラム開発にも積極的に使われるようになりました。ただし、コンパイラによって仕様（使い方）が異なるという言語であったことから、1989年に**ANSI**※（米国国内規格協会）が標準規格を定め、さらに1990年には**ISO**※（国際標準化機構）により規格が制定されました。

このように、C言語は使いやすいものになっていったのです。

C言語の特徴

C言語というプログラミング言語は、**高級言語**（人間に理解しやすいほど高級といわれる）の機能と、**アセンブラ**（コンピュータに理解しやすい言語で、機械語に近い言語）に近い機能とを合わせ持っている言語で、言語自体が単純で学びやすいものとなっています。そのため、文の記述を短く簡潔にすることもできます。

また、多くの演算子を用意しているので、複雑な計算式を簡単に書くこともできますし、アセンブラに近い機能を持っているので、コンピュータをきめ細かく制御できます。そして、便利なのは、古いコンピュータでも新しいコンピュータでも動きますし、多くのOSでも使えるということです。

C言語の特徴をまとめると、次のようになります。まだ、わからない言葉が出てくるかもしれませんが、あとで説明しますので安心してください。

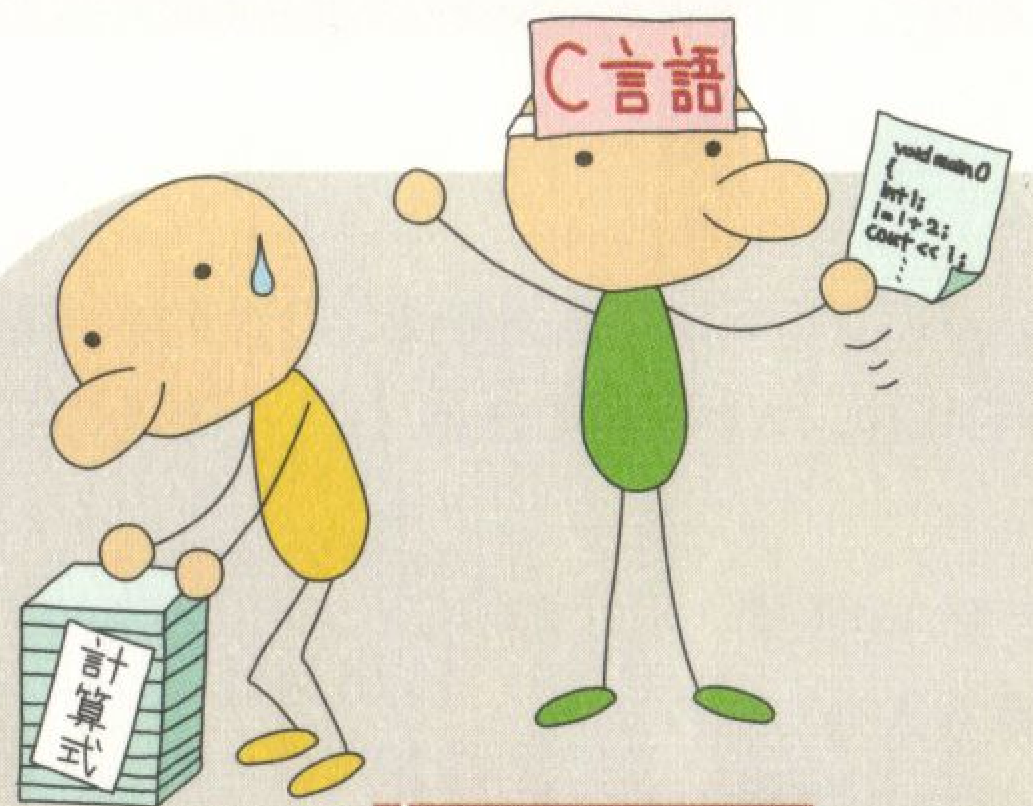
- 自分なりの書き方ができます。
- 順次処理、繰り返し、条件分岐という3つの制御構文が用意されています。
- 文の記述を短く、簡潔にすることができます。
- プログラムが関数の集まりからできているので、いろんな機能を関数としてまとめることができます。
- 変数や関数のアドレスを取り扱うことができます。
- 言語自体が単純で学びやすくできています。

※ANSI　American National Standard Instituteの略。「アンシー」と読む。
※ISO　International Standard Organizationの略。「アイエスオー」「イソ」とも読む。

- 複雑な計算式が簡単に書けます。
- コンピュータをきめ細かく制御できます。
- どんなプログラムでも作ることができます。
- コンピュータの世界での共通言語です。



言語自体が単純で学びやすい！

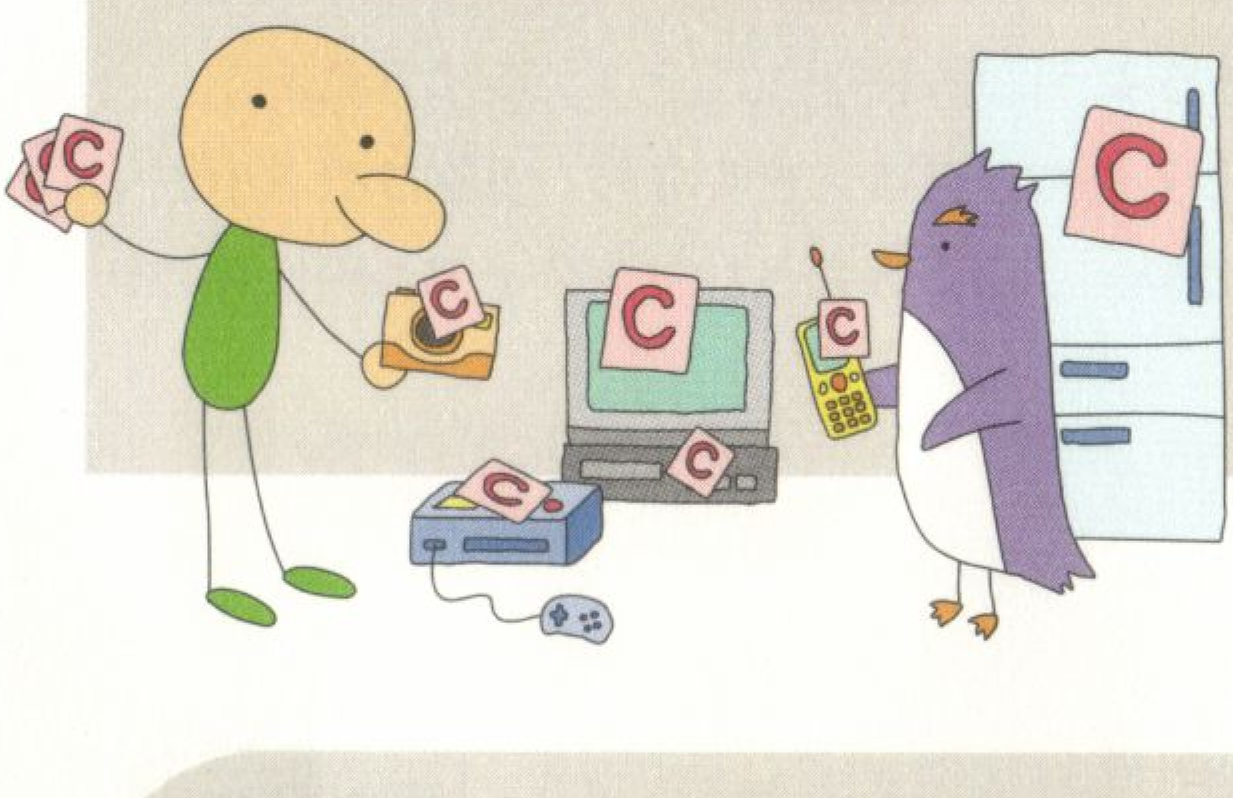


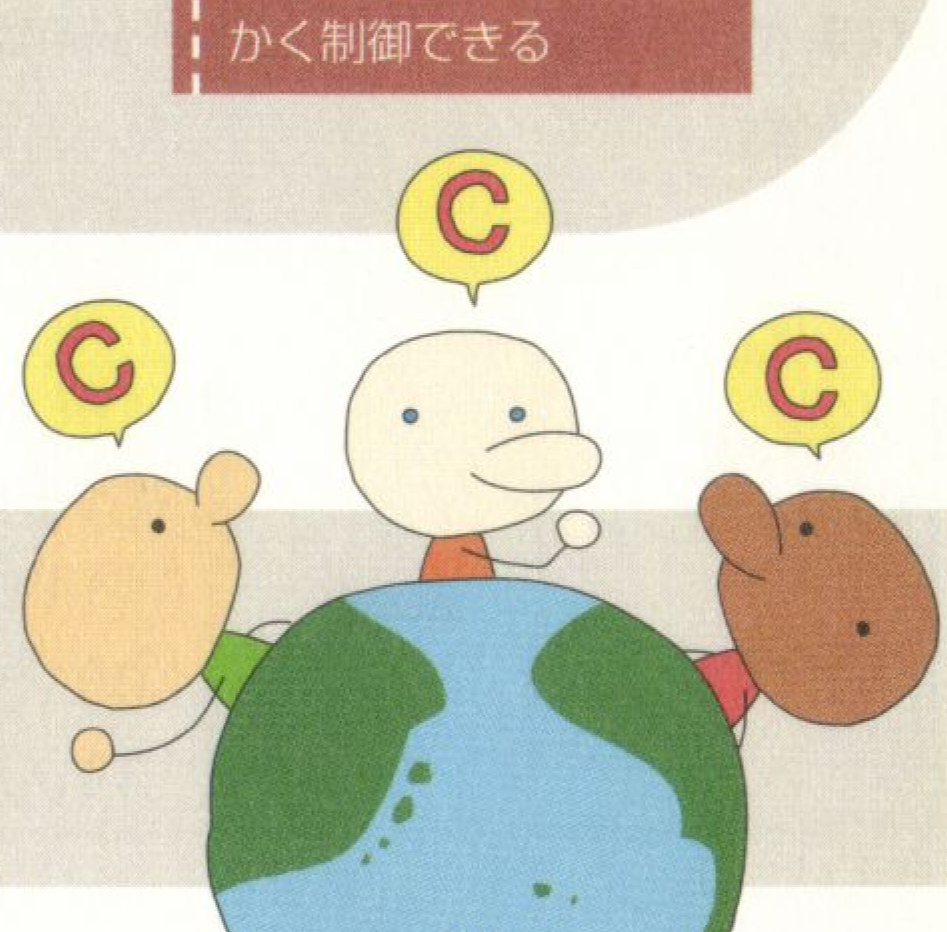
複雑な計算式が簡単に書ける



コンピュータをきめ細かく制御できる

どんなプログラムでも作ることができる





コンピュータの世界での共通言語

Section 02

C言語のプログラム

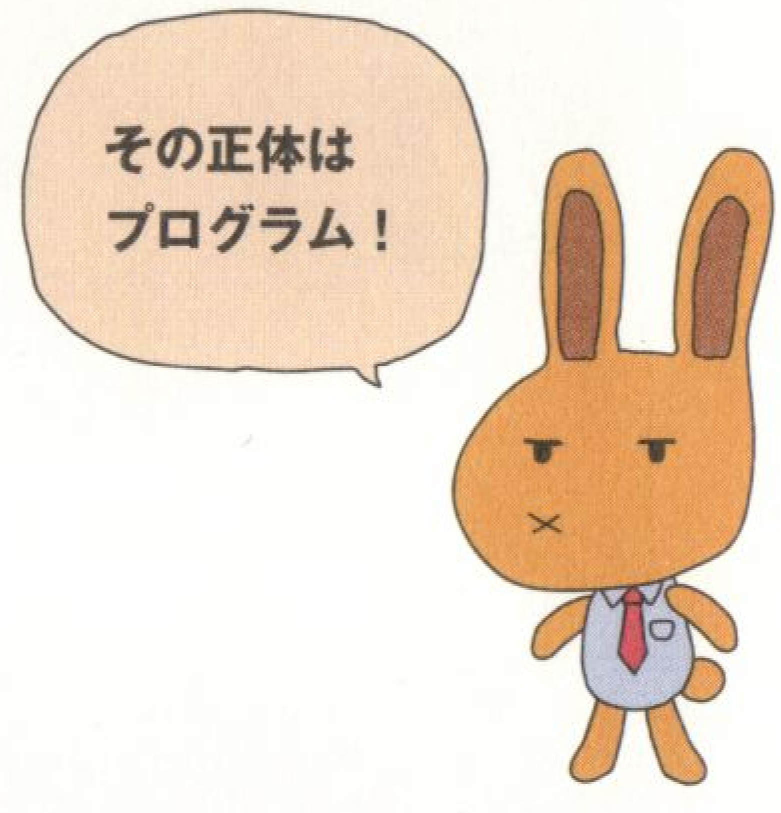
プログラムとは何でしょうか？　ここではプログラムの仕組みから、インタプリタとコンパイラについて、説明していきます。

プログラムとは

コンピュータにプログラムするためには、プログラミング言語を使うのですが、具体的にはどうするのでしょうか？　言語というからには、人が理解できる「文」があるのです。

プログラムとは、この文の集まりのことををいいます。

例えば、パソコンのワープロソフトで文章を入力したり、携帯電話でメールをやり取りしたりするとき、その正体はすべてコンピュータが理解できるように作られたプログラムと呼ばれるものになるわけです。



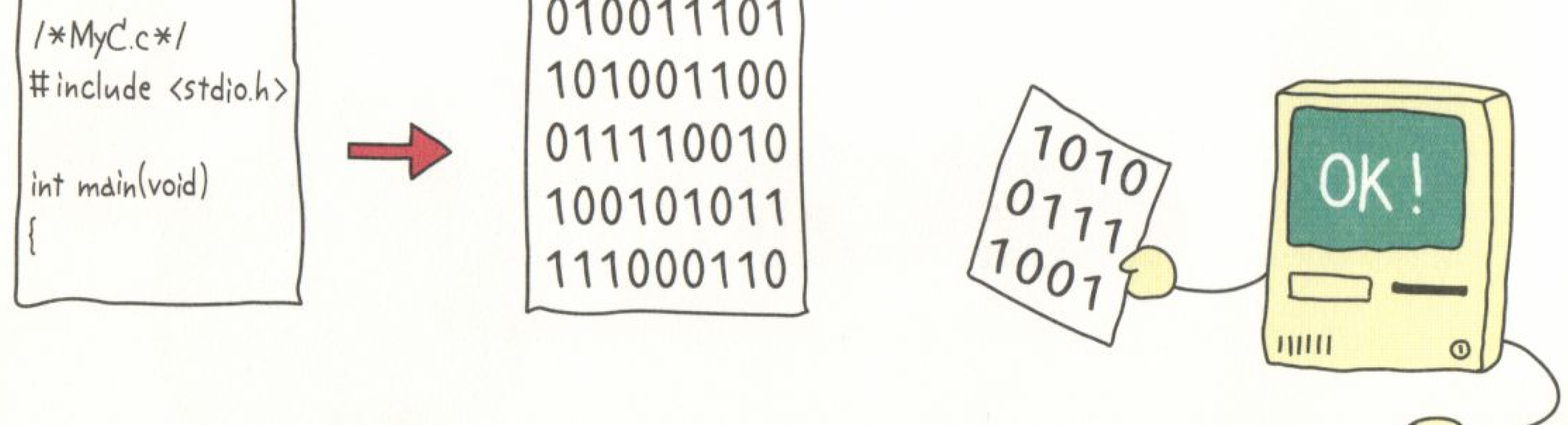
では、プログラムのもとになっているのは何なのでしょうか？

それは、人がもっと理解できるように作られた文の集まりである**ソースコード**というものなのです。

ソースコードは、コンピュータが理解できるように最終的には0と１に変換されます。

また、この文の集まりが多ければプログラムのサイズは大きくなり、少なければ小さくなります。ちなみに、コンピュータの世界では、プログラムとソフトウェアは同じものを指します。

最終的に0と1に変換されて
コンピュータは理解する



プログラムの仕組み

これから、C言語で作成できるプログラムについて説明していきますが、その前にプログラムの仕組みについて考えてみましょう。

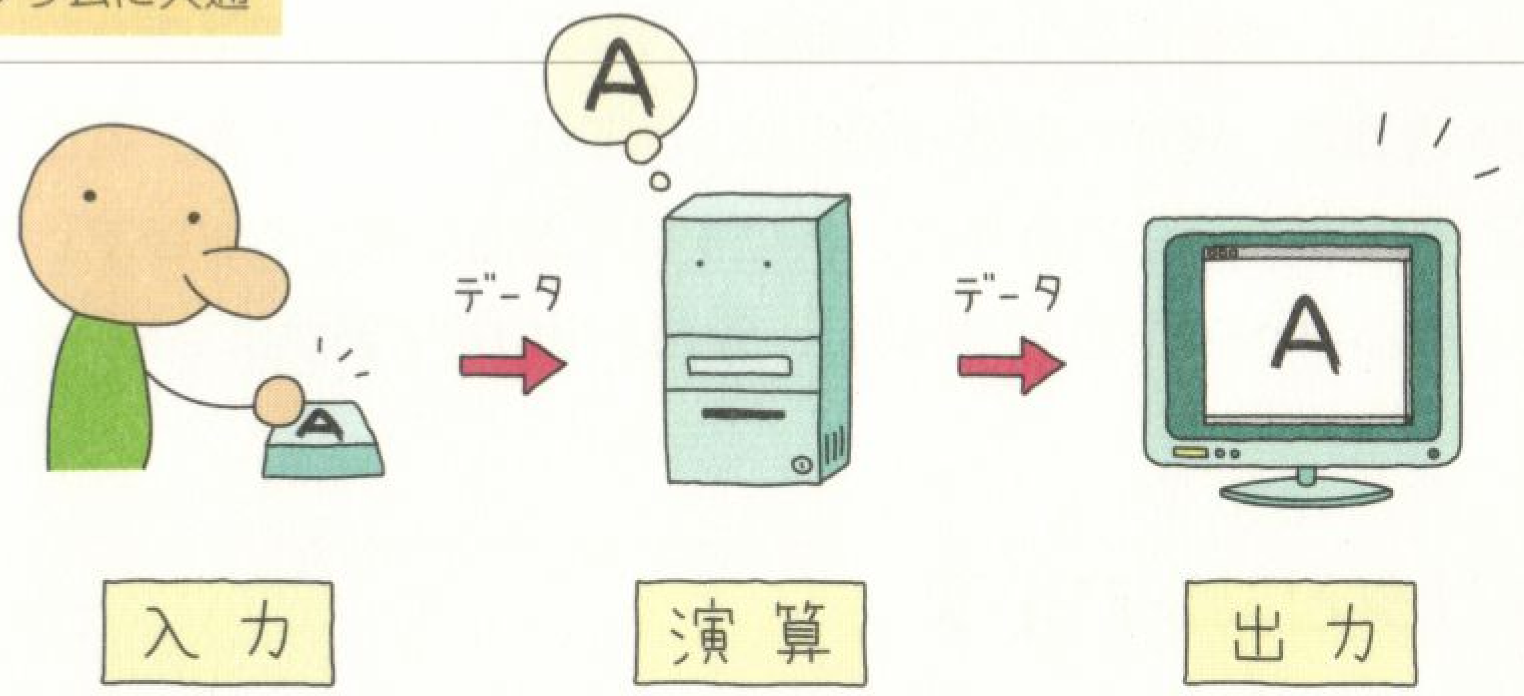
例えば、ワープロソフトのプログラムは、何をしているのでしょうか？

ワープロソフトは、キーボードから入力された文字を画面に表示したり、プリンタによる印刷をします。

もう少し詳しく説明すると、キーボードを通して入力された文字を、データとして受け取ったワープロソフトは、コンピュータに画面や用紙の適切な位置に表示する計算（演算）を指示して、その結果を画面やプリンタを通して出力させているということになります。

この一連の流れ、**入力➡演算➡出力**は、ワープロソフトのプログラムだけでなく、すべてのプログラムにおいて共通です。プログラムの仕組みは、基本的にどれも同じパターンなのです。

すべてのプログラムに共通



▶▶2進数について

0と1だけの数字の世界を**2進数**といいます。

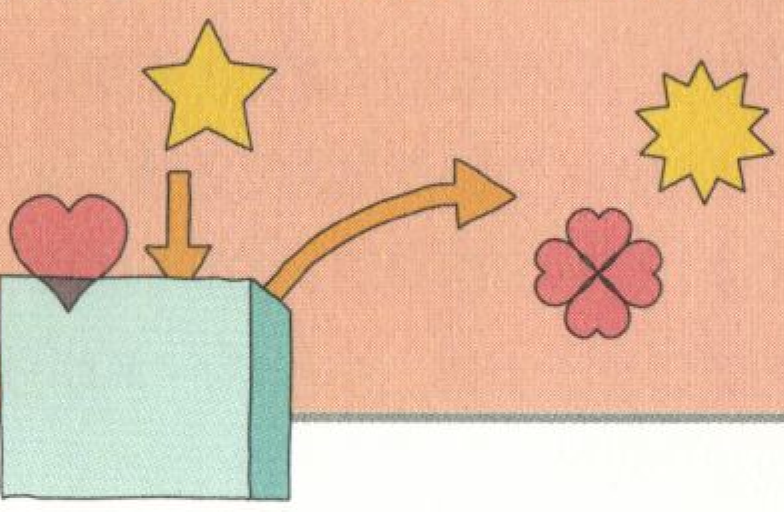
コンピューターは、2進数によってプログラムを処理します。

コンピュータが、なぜ2進数で処理するのかを説明すると、コンピュータは、電流が流れる（電圧がかかる…1）か流れない（電圧がかからない…0）かを基本として構成されていることもありますが、効率よく数を扱うには、最低でも0と1があれば問題がないからです。



例題　入力と出力

何かを入力したら何かを出力するようなものを考えてみてください。



●解答

例．体重計

●解説

人が体重計に乗ります（入力）。体重計が計算します（演算）。体重の結果が表示されます（出力）。

例えば、自動販売機ではどうでしょうか。自動販売機も、入力（お金）と出力（モノ）で表せます。しかし、厳密にいうと、自動販売機は入力と出力では表せないのです。なぜならば、自動販売機は、確かにある一時点においては、入力（お金）と出力（モノ）で表せますが、実際はモノが出されたあと、またすぐに入力を待っています。このように、半永久的に動いているようなものは、単純に、入力と出力では表せません。

インタプリタとコンパイラ

高級言語で作成されたソースコード（おおもとになるプログラムの文字列）は、**インタプリタ**、または**コンパイラ**を用いて機械語に翻訳されます。

インタプリタは、ソースコードを１ステップずつ解釈しながら実行していきます。

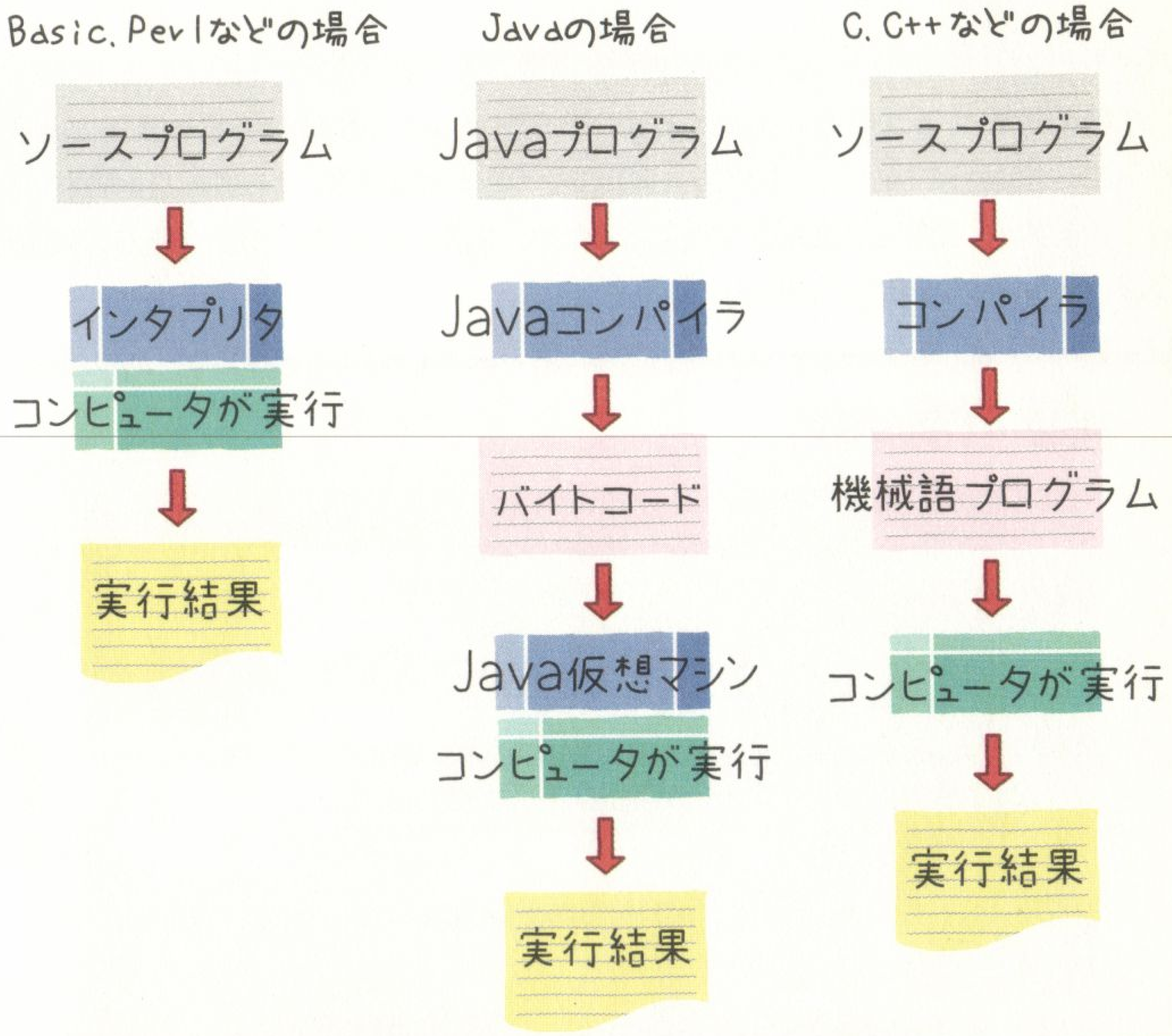
それに対してコンパイラは、すべてのソースコードを一度に機械語に翻訳してプログラムを作ります。作成されたプログラムは、必要なときにユーザーが実行します。翻訳と実行が一度に実行されるので、インタプリタのほうが手軽にプログラミングすることができます。しかし一般的には、コンパイラによって機械語に翻訳されたプログラムのほうが、インタプリタによる実行より処理が速くなります。

▼インタプリタとコンパイラの特徴

インタプリタ	1行ずつ翻訳して、1行ずつ実行します。翻訳しながら実行するので処理が遅いですが、反面、プログラムの間違いが見つけやすくなります。
コンパイラ	すべてのソースコードをまとめて翻訳します。そのため処理は速いですが、プログラムの間違いは見つけにくくなります。

インタプリタとコンパイラの違いは、英語から日本語への翻訳にたとえることができます。インタプリタは、一文一文を話した直後に日本語へ翻訳しますが、コンパイラは、英語で全文を話し終わったあと、その内容をまとめて翻訳するというものだと考えてください。

インタプリタ系の言語としては「Basic」や「Perl」、コンパイラ系の言語としては「C」「C++」などがあげられます。ちなみに「Java」は、バイトコード（Javaコンパイラによって翻訳されたコンピュータの理解できるコード）へのコンパイルが必要な特殊なインタプリタ言語です。



▶▶C言語と文字コード

ここで、C言語と文字コードについて簡単に説明しておきましょう。

コンピュータは、文字を扱うとき、人のように文字を理解しているわけではありません。コンピュータは、文字を「0と1からなる」数値の列として扱います。この数値の列のことを**文字コード**といいます。文字コードには様々な種類があります。よく知られているものをあげると、英数字を扱う文字コードとしてASCII※（アスキー）コードが、日本語の文字を扱う文字コードとしてShift-JIS※（シフトジス）が、アジアの言葉を扱う文字コードにはEUC※（イーユーシー）などがあります。

C言語が扱う文字コードは、**Unicode**※（ユニコード）というものですが、これは、世界共通の文字コードです。コンピュータの世界は欧米が起源になっているので、これまでアルファベットや数字などは1バイトコード、漢字やかなは2バイトコードなどとなっていて、英語圏で作られたソフトウェアを日本語化するのをいつも難しくしていました。それに対してUnicodeは、アルファベットも、数字も、漢字も、かなも、とにかく全部2バイトコードとして扱うので、言語の違いによる修正が不要になります。

▶専門的な説明

バイトとは8つのビット（bit）を集めた情報量を表す単位です。1ビットは「0か1か」の2種類の情報を表すので、1バイトは2の8乗 = 256種類の情報を表現できます（00000000から11111111まで）。アルファベットや数字は、1バイトで全部の文字を表現できますが、漢字は7000文字以上あるため、2バイト=2の16乗（6万5536種類）で表現します（本文28ページ参照）。

COLUMN C言語の歴史

◇◇◇1960年代◇◇◇	
1969	AT&TのベルA研究所でUNIX（UNICS）が誕生
◇◇◇1970年代◇◇◇	
1970	C言語の元となるB言語が開発される
1973	カーニハンとリッチーによってがC言語が開発される
1978	C言語のバイブル「The C Programming Language」が出版される
1979	ストラウストラップによってオブジェクト指向をサポートしたC++が開発される
◇◇◇1980年代◇◇◇	
1989	米国国内規格協会のANCI-Cが制定される
◇◇◇1990年代◇◇◇	
1990	国際標準化機構のISO C言語が制定される
1993	国内でJIS C言語が制定される
1999	改訂版標準Cが制定される

Linuxの元ともなっているポピュラーなOS(オペレーティングシステム)です。

「B」言語の次だから「C」言語というわけです。

UNIXはその後C言語によって記述し直されました。

K&Rという愛称で親しまれています。

※ASCII　American Standard Code for Information Interchangeの略。1963年ANSI（米国規格協会）により制定。ASCIIコードはコンピュータ用の英数字コードとして最も普及している。
※Shift-JIS　日本語環境で広く利用されている日本語文字コードの通称。
※EUC　拡張UNIXコード。UNIXというOS（オペレーティングシステム）のアジア言語サポートとして採用されている文字コード。
※Unicode　単一の2バイトの文字コードによって世界の文字の多くを表現しようとする文字コード。中国語でも日本語でも、同一とみなしうる漢字には同じコードを割り当てるなどのUnificationを行っている。

COLUMN あなたのプログラマー度

YES/NOで簡単にわかる

START!

YES

NO

1 ロールプレイングゲームが大好きだ。

2 ものごとは分析的に考えてから決める。

3 直感で判断するほうだ。

4 算数が得意である。

5 わからないことは友達に聞く。

6 何でも新しいものが好きである。

7 妄想をしてしまう。

8 他人に対して素直に謝ることができる。

9 何でも分解したい。

10 効率よく仕事を片付けることができる。

11 自分の技術に関心を持っている。

12 完璧でないと、気が済まない性格だ。

A

あなたのプログラマー度は

達人

あなたが嘘をつくことなくここにたどりついたのなら、必ずや達人クラスのプログラマーになれることでしょう。あとは妄想のしすぎに注意するだけです。

B

あなたのプログラマー度は

職人

頑固一徹のあなたは、職人としてのプログラマーを目指しましょう。1つのことを深く追求して、得意分野を持てば、おのずと道は開けるに違いありません。

C

あなたのプログラマー度は

凡人

凡人に選ばれてしまったあなた、がっかりしなくても大丈夫です。プログラマーの大半は、凡人なのです。地道な努力をしていれば、いつかは報われます。

practice 練習問題 解答は213ページ

1 プログラミング言語 難易度★☆☆☆☆

C言語以外のプログラミング言語には、どんなものがあるのかを調べてみましょう。

▶▶ヒント：本文17ページ参照

2 プログラミング言語 難易度★★★☆☆

プログラミング言語を、大きく3つの種類に分けたとき、どのような種類に分けられるのかを答えてください。

▶▶ヒント：本文17ページ参照

3 プログラミング言語 難易度★★★★★

インタプリタとコンパイラの、共通の役目について、説明してください。

▶▶ヒント：本文22ページ参照

memo

Chapter1のまとめ

★プログラミング言語

プログラム言語とは、プログラムを開発するために用意された人工の言語のことです。

★プログラム

プログラムとは、プログラミング言語の文の集まりで、コンピュータへの指示を書いたものです。

★プログラムの仕組み

プログラムは、「入力➡演算➡出力」という流れを持っています。

★アプリケーション

アプリケーションとは、コンピュータを使って、文書や表、データベースを作るなど､それぞれの目的を実現するためのプログラムのことです。

★インタプリタとコンパイラの役目

インタプリタとコンパイラの役目は、人間がわかりやすいプログラム（ソースコード）から、機械がわかりやすいプログラム（命令）に変換することです。

★インタプリタとコンパイラの違い

インタプリタとコンパイラの違いは、1行1行変換して実行するのが、インタプリタ、一括して変換するのが、コンパイラです。

★C言語の特徴

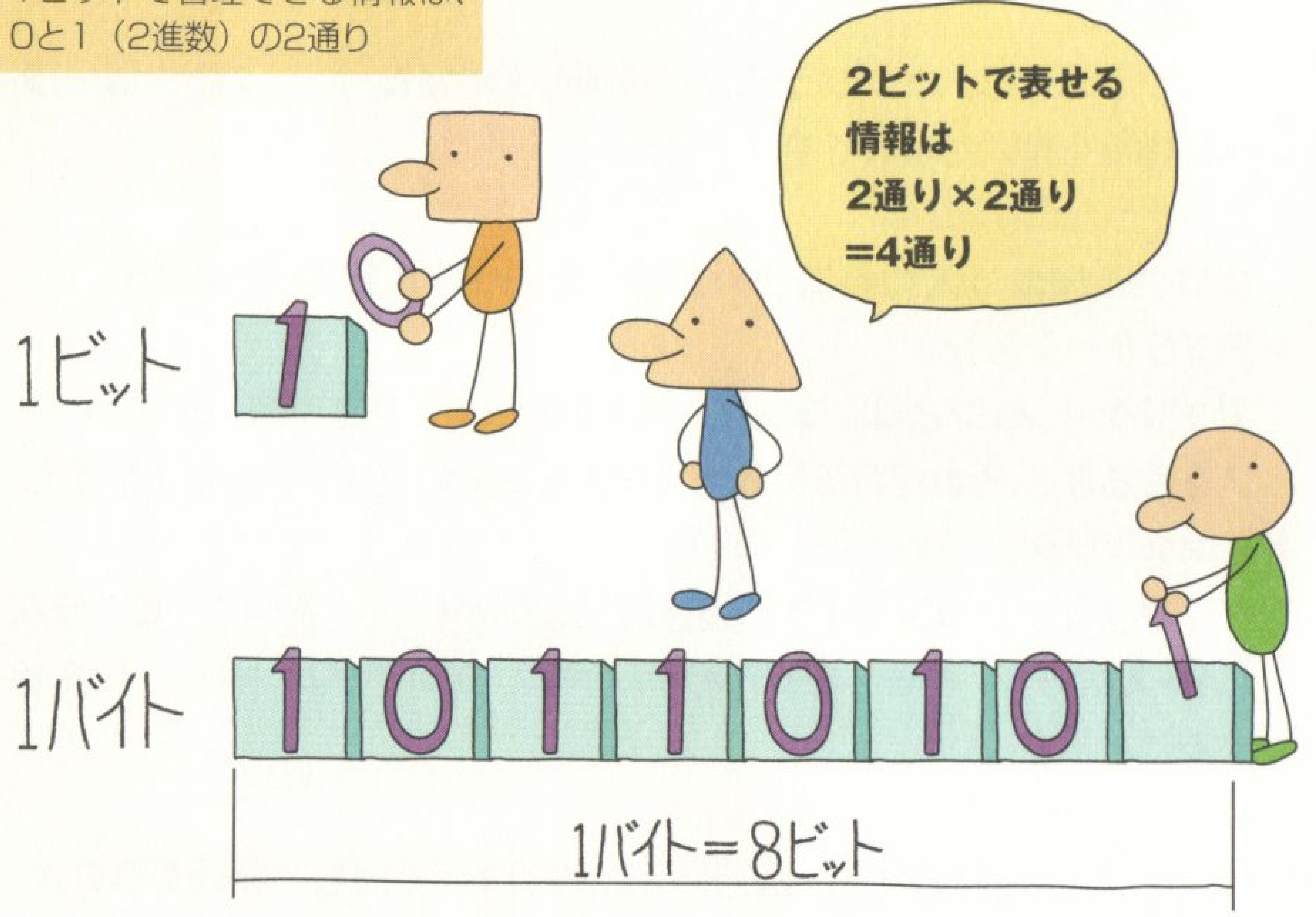
- 言語自体が単純で学びやすいです。
- 複雑な計算式が簡単に書けます。
- コンピュータをきめ細かく制御できます。
- どんなプログラムでも作ることができます。
- コンピュータの世界での共通語です。

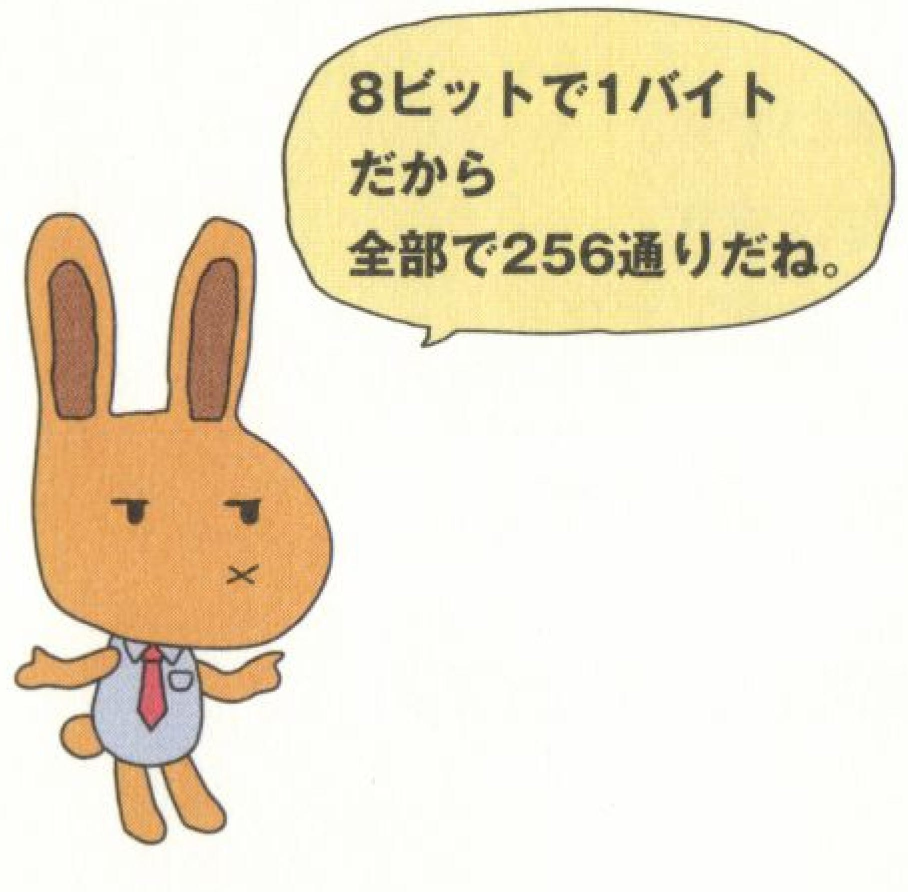
★Unicode（ユニコード）

Unicodeとは、C言語が扱う文字コードのことです。

COLUMN ビットとバイトは基本だ！

1ビットで管理できる情報は、
0と1（2進数）の2通り

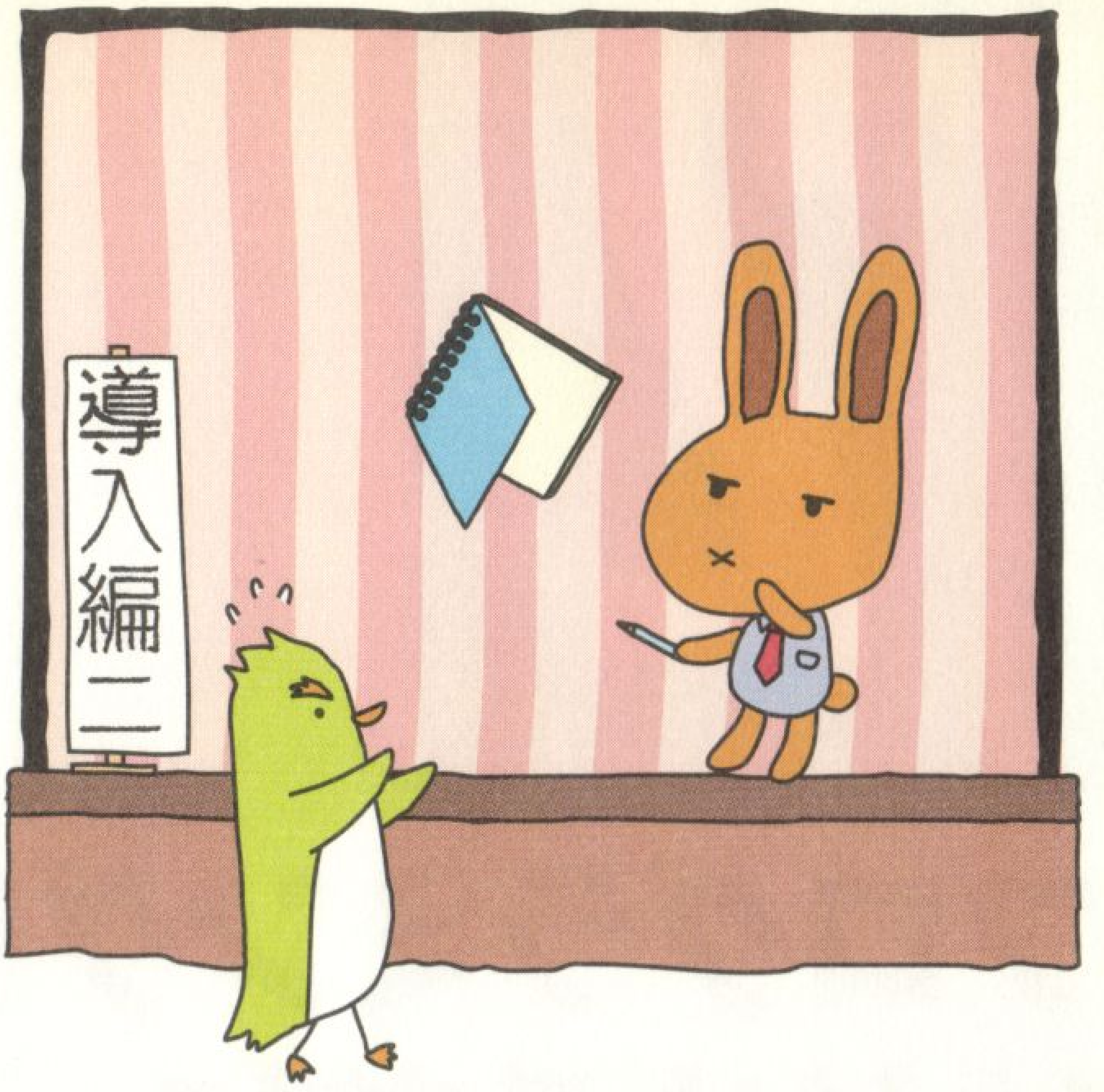




1バイトで表現できる情報量は
2×2×2×2×2×2×2×2
$=2^8$
=256通り

「0か1か」の、1つのまとまりを**ビット**といいます。つまり、1ビットで2つの状態を表すことができます。

これを2つ（2ビット）使えば、4つの、3つ（3ビット）使えば、8つの状態を表すことができます。8ビットのことを、1バイトといいます。



★C言語プログラミングの準備をします。
準備ができていない人は、ページをめくってください。
「gcc」をインストールして、

実際にプログラムを試してみましょう。

★すでにプログラミングの準備ができていて、
C言語の基本を学びたい方は

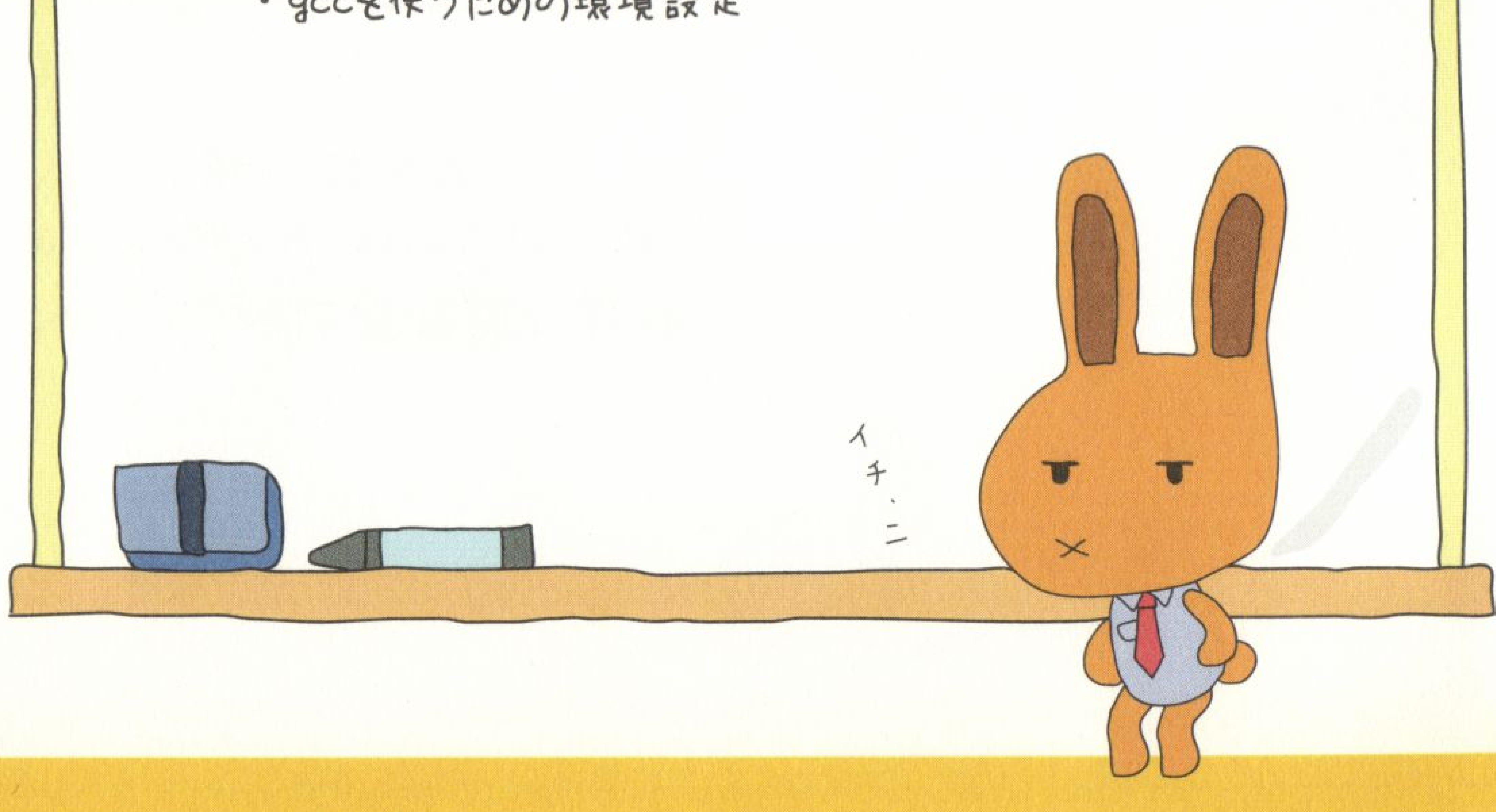
➡第3章本文54ページへ

★C言語の基本を理解していて、
実際のプログラミングから学びたい方は

➡第4章本文72ページへ

Chapter 2

プログラミングをはじめる前に



この章でマスターすること

作文を書く前に、鉛筆や原稿用紙、消しゴムなどが必要になるように、C言語でプログラミングをはじめるにあたっても、必要なものを準備しなければなりません。

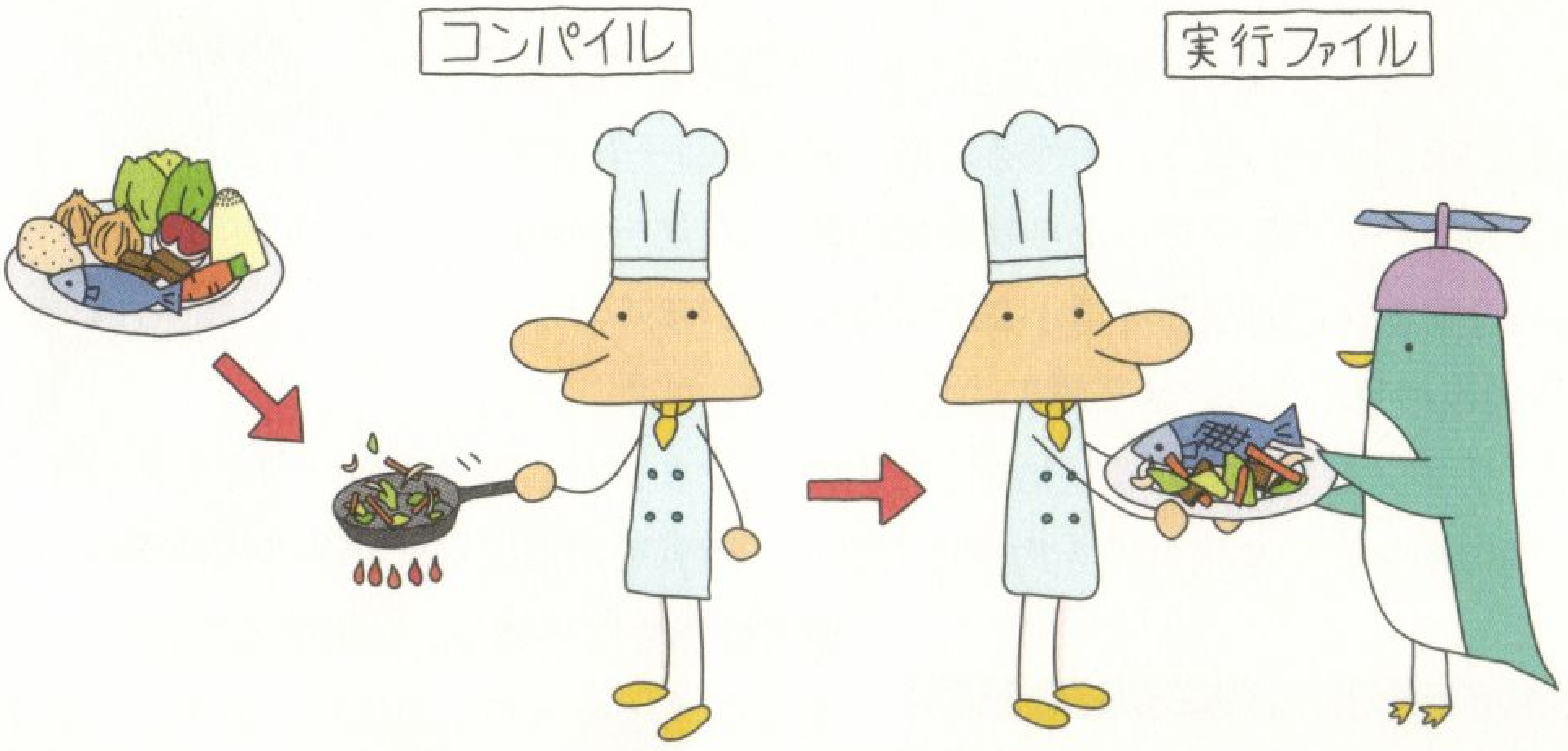
ソースコードを書いたり、保存するために必要な**テキストエディタ**、キーボードから文字を入力してコンピュータを操作する**コマンドライン**、そして、C言語の**コンパイラ**です。



プログラミングをおぼえるには、とにかく経験して、実感することが大切です。これらの準備ができたら、まずは、C言語のプログラムのサンプルを動かしてみましょう。

それでは、わからないことはさておき、とりあえずソースコードを書いて、コンパイルして、実行してみることにしましょう。

コンパイルすると実行ファイルになる？



C言語必須用語

テキストエディタ　gcc
コマンドライン　パス

ステップアップ用語

CUI
GUI

Section 01

開発ツールを準備しよう

C言語でプログラミングをはじめるにあたって、必要なものを最低限3つ準備しなければなりません。プログラマーになるためには、これらは必要な道具ですし、それを何のために使うのかも知っていなければなりません。ここでは、それを説明していきましょう。

プログラミングへの心構え

C言語をおぼえる近道は、とにかく誰かの書いたソースコードを見て、自分で書いて、実行してみることです。

C言語を使うということは、コンピュータとコミュニケーションすることであって、そこにコンピュータがなければ意味がありません。そして、いちばん重要なのはコンピュータとコミュニケーションすることや、C言語によって何かを作る楽しみを見出すことができることなのです。

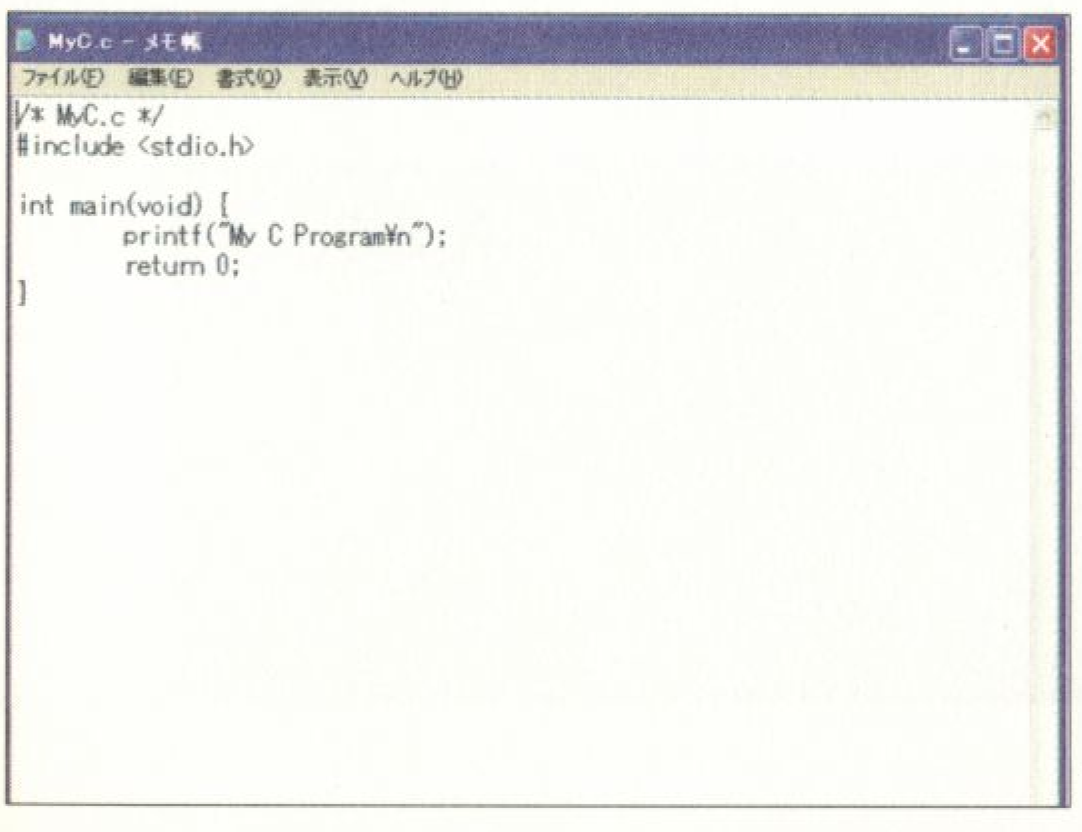
テキストエディタ



コンピュータを使って書く、簡単な文書やソースコードなどを、**テキスト**といいます。プログラムを作るには、ソースコード（テキスト）を書いて、それを保存したりするためのアプリケーションが必要になります。これを**テキストエディタ**といいます。

Windowsの場合であれば、**メモ帳**を使えばよいでしょう。

ところで、テキストを入力するためのアプリケーションとしては、ワープロもありますが、ワープロは単なるテキストの入力だけでなく、印刷を前提にした文書作成の機能（文書の整形や文字飾り、印刷形式の設定など）をたくさん持っている点が、テキストエディタと違うところです。




```
/* MyC.c */
#include <stdio.h>

int main(void) {
	printf("My C Program¥n");
	return 0;
}
```

◀メモ帳

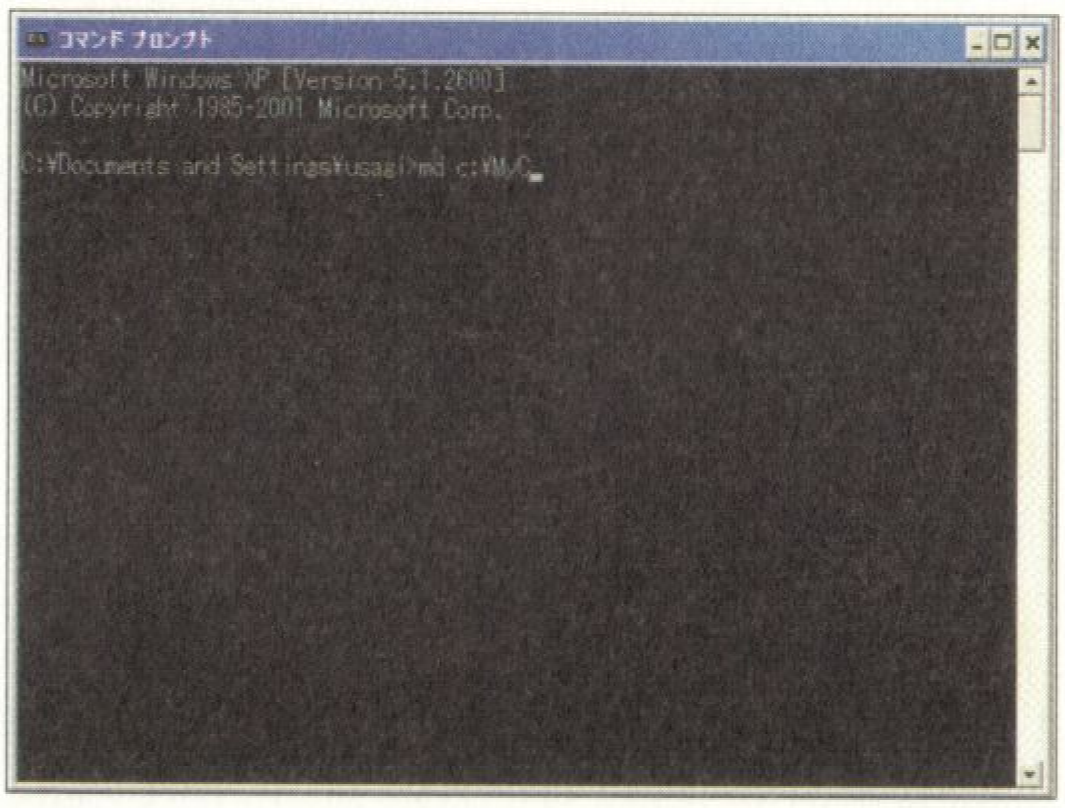
Windows XPの場合

スタートメニュー➡すべてのプログラム➡アクセサリ➡メモ帳

コマンドライン

コンピュータに、プログラムの作成や実行などの簡単な指示を伝えるには、**コマンドライン**というものを必要とします。コマンドラインとは、ユーザーがコマンドを入力するための行のことをいいます。

コマンドラインは、キーボードから文字を入力することによって、コンピュータを操作することができます。Windows XPの場合であれば、**コマンドプロンプト**を使えばよいでしょう。



◀コマンドプロンプト

Windows XPの場合

スタートメニュー ➡ すべてのプログラム ➡アクセサリ➡ コマンドプロンプト

例題 コマンドプロンプトを使ってみよう

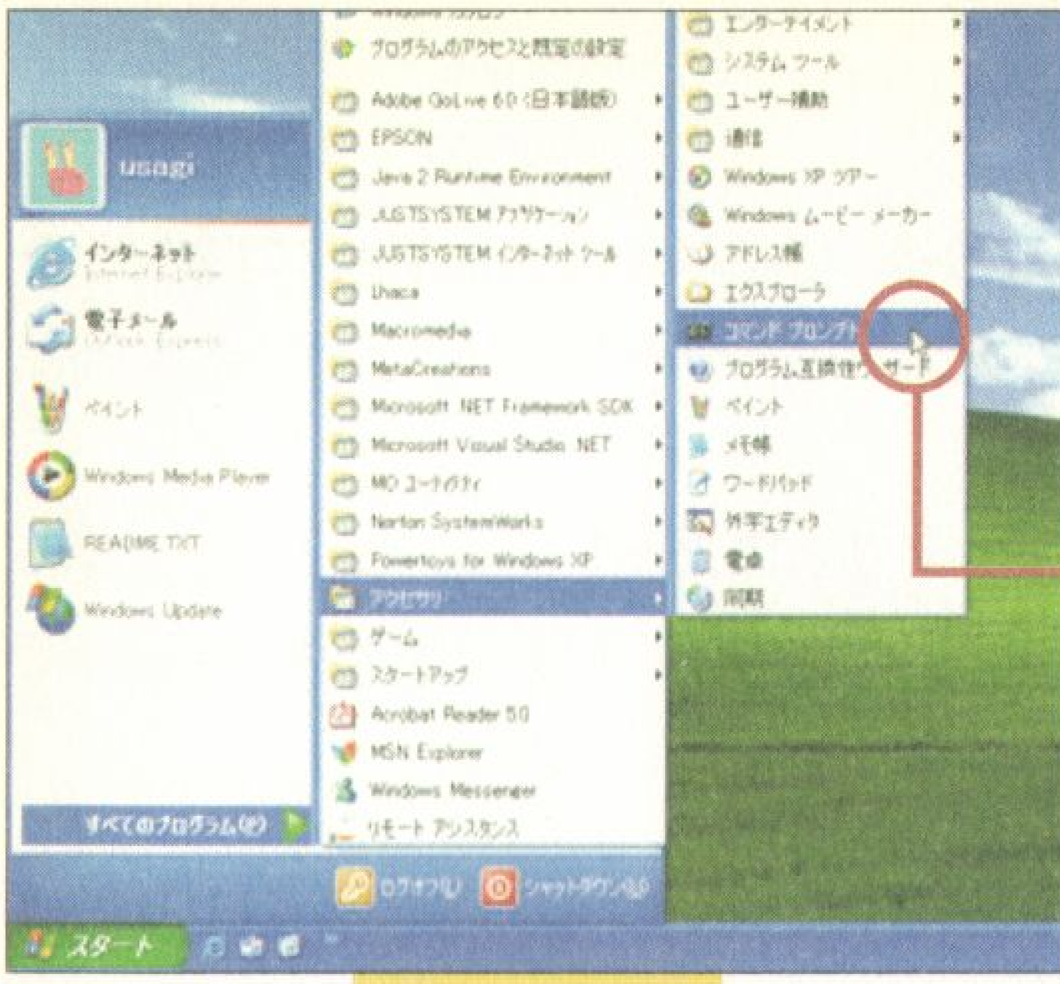
「dir」コマンドで、ファイルを表示してみましょう。

●解説

コマンドプロンプトでは、フォルダのことを**ディレクトリ**（**Directory**）といいます。Windowsでは、エクスプローラなどでファイルの一覧を見ることができますが、コマンドプロンプトでファイルの一覧を見るには**dir**コマンドを使います（「dir」とコマンドプロンプトに入力する）。

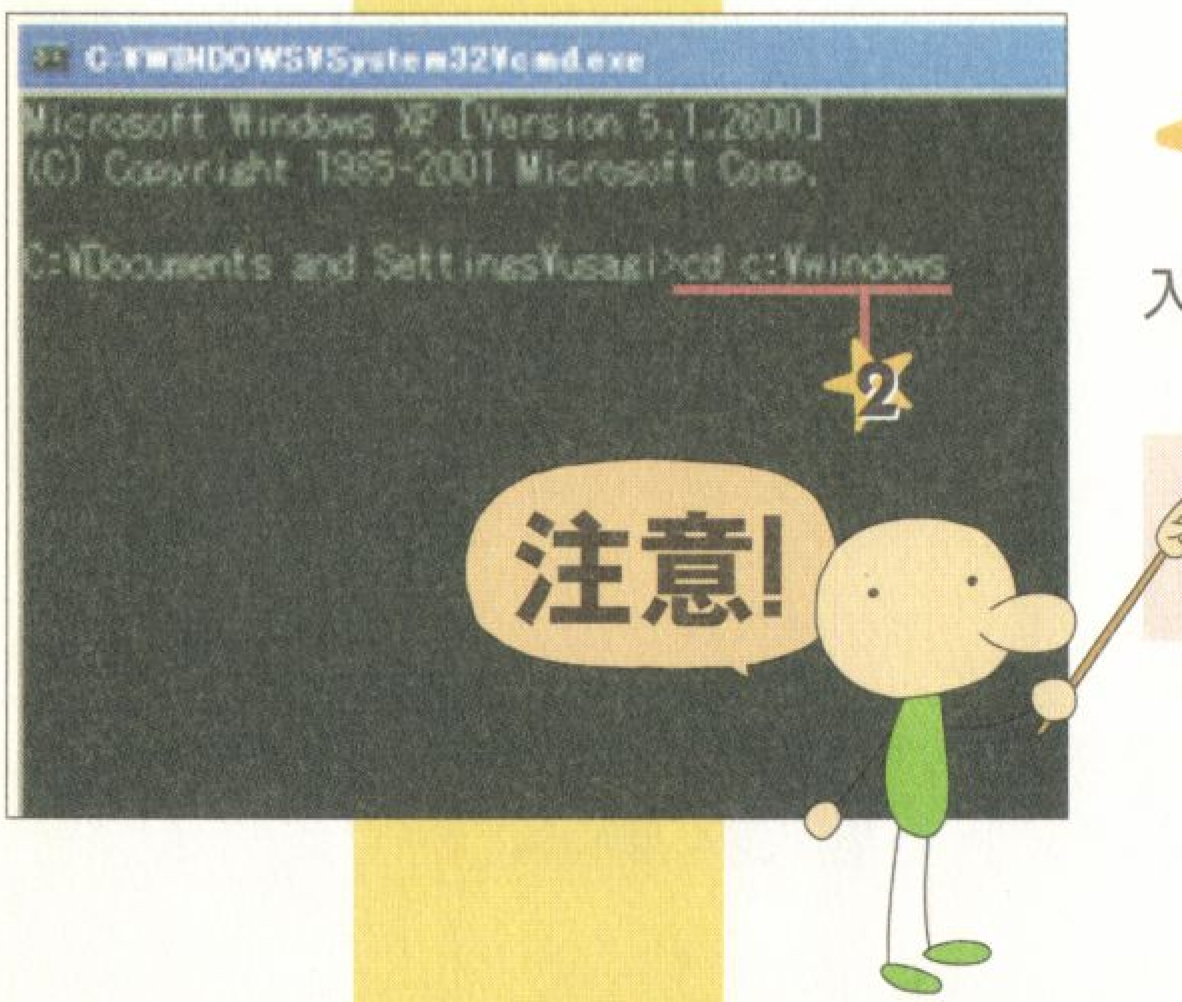
このように、キーボードから文字を入力して、結果を文字で受け取るような、コンピュータとのコミュニケーションを図る手段のことを、**CUI** (Character-based User Interface) といいます。一方、アイコンやウィンドウといった「図的」なものを使って、コンピュータとのコミュニケーションを図る手段のことを、**GUI** (Graphical User Interface) といいます。

それでは、次のようにしてファイルを表示してみましょう。



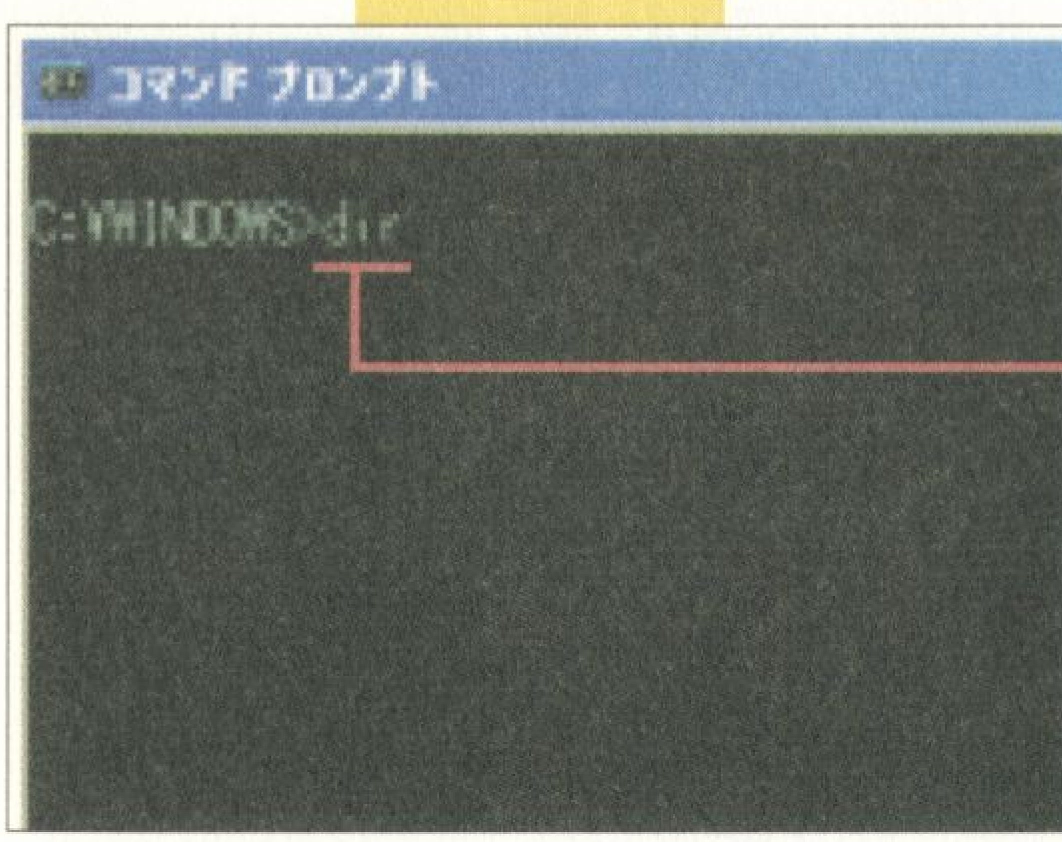
1 コマンドプロンプトを起動します。**スタートメニュー➡すべてのプログラム➡アクセサリ➡コマンドプロンプト**をクリックしてください。

1 コマンドプロンプトをクリックする。



2 **コマンドプロンプト**に、「**cd c:¥windows** [Enter]」と入力します。

[Enter] は「Enter」と入力するのではなく、Enterキーを押してください。



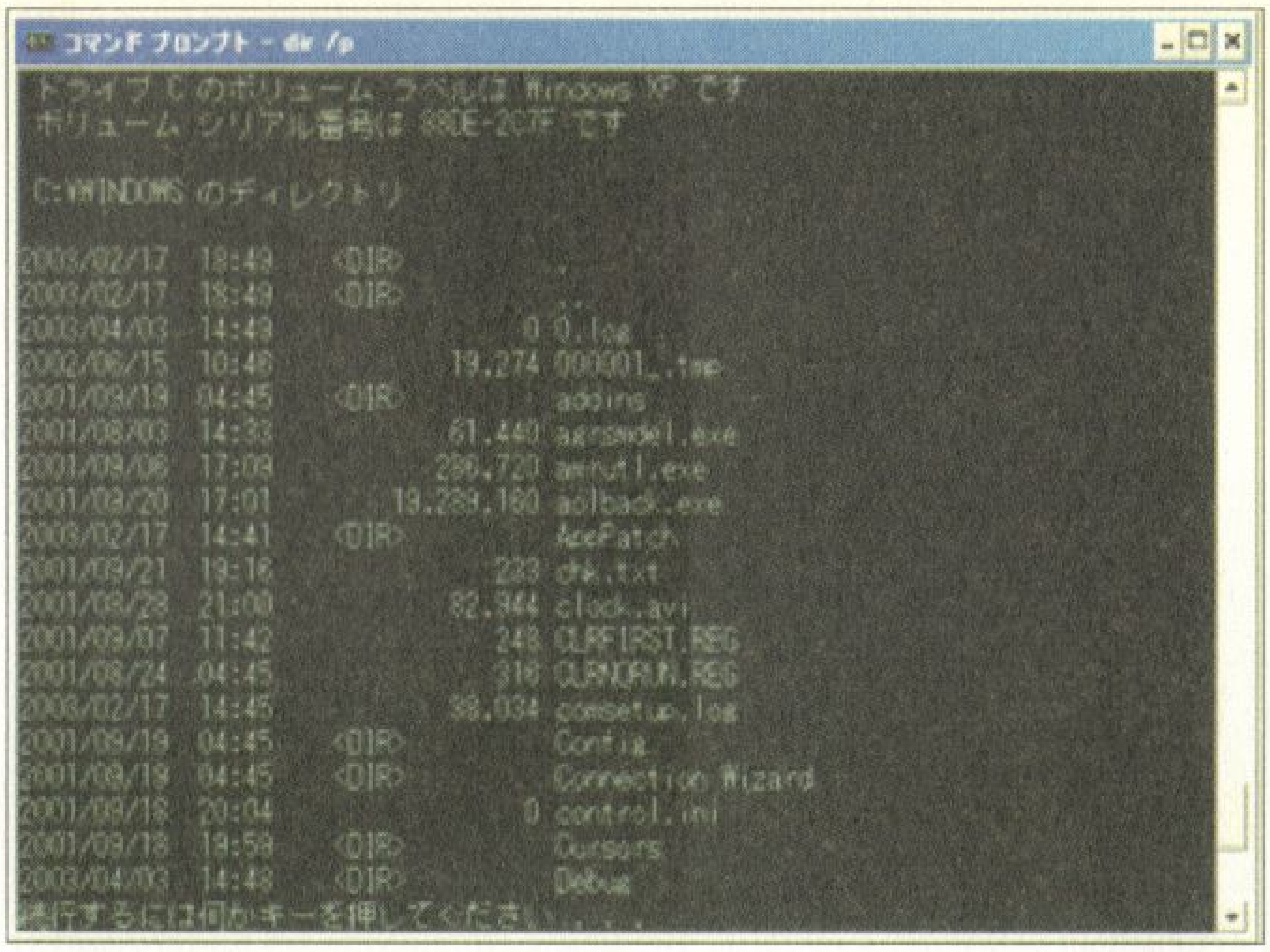
3 **コマンドプロンプト**に、「**dir** [Enter]」と入力します。

3 dirと入力する。

どうですか、ファイルの一覧が出てきましたか？

ところで、あっという間にスクロールしてよくわからなかった場合もあるかもしれません。そういうときは、3 の「dir」のところを、**dir /p**と入力してみましょう。

この「/p」は、**オプションスイッチ**というもので、コマンドの動作状態を指定します。



「/p」は、「ポーズ（pause）」の意味で、ビデオやDVDの一時停止と同じように、1画面ごとに表示を停止するというものです。

▶専門的な説明

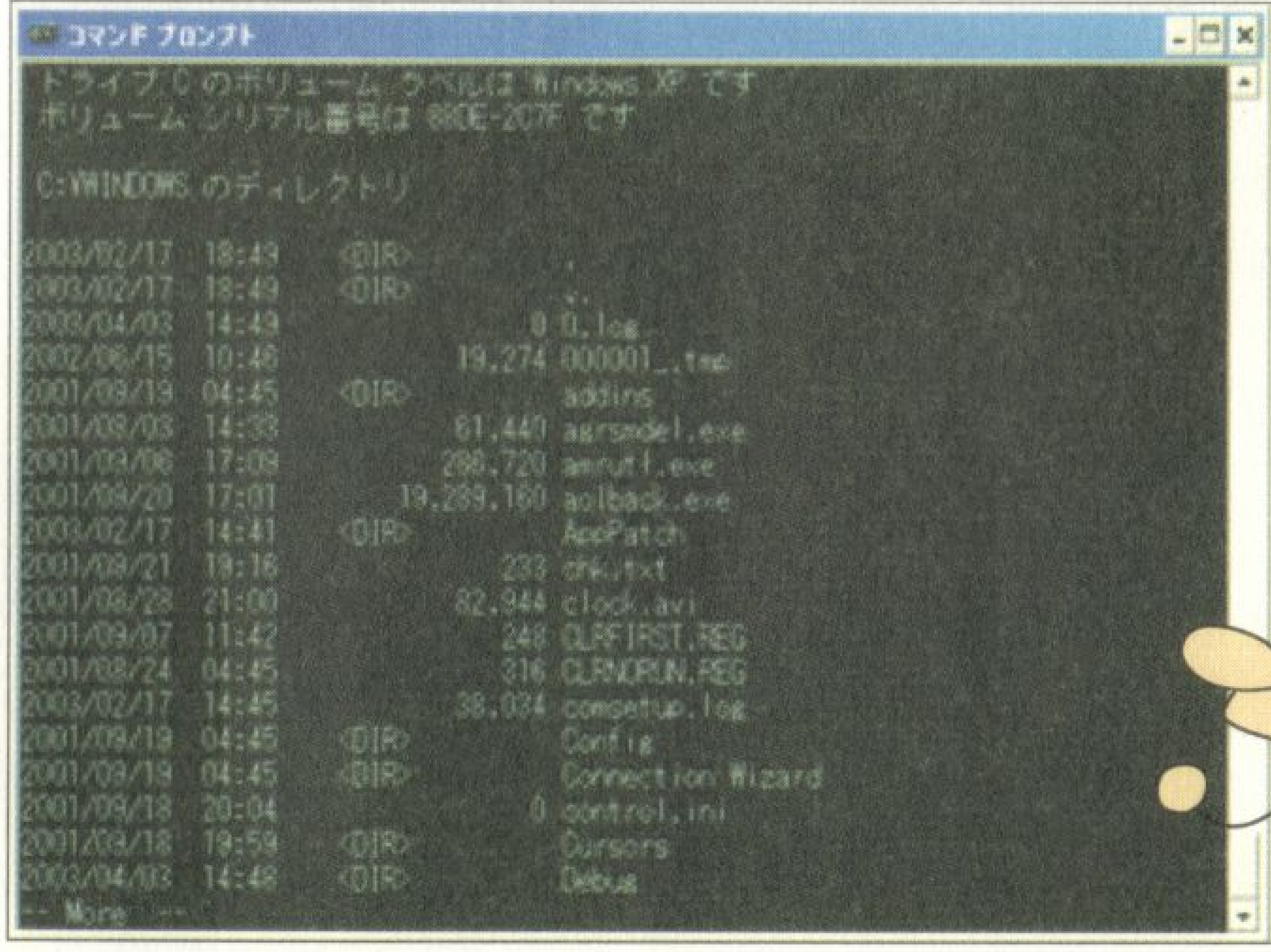
ところで、あっという間にスクロールしてよくわからなかった場合に、画面を一時停止させるには、次のような方法もあります。

3 の「dir」のところを、「dir｜more」と入れてみてください。

「｜more」を末尾に付けると表示内容を、Enter キーを押すごとに、1行ずつに区切ることができます。

先ほどの「/p」オプションとの違いは、「/p」オプションは、dirコマンドが自分で1画面ごとに区切って表示するのに対し、「｜more」は、dirコマンドの表示結果をコマンドプロンプトが横取りして、1行ごとに区切って表示させている点です 。

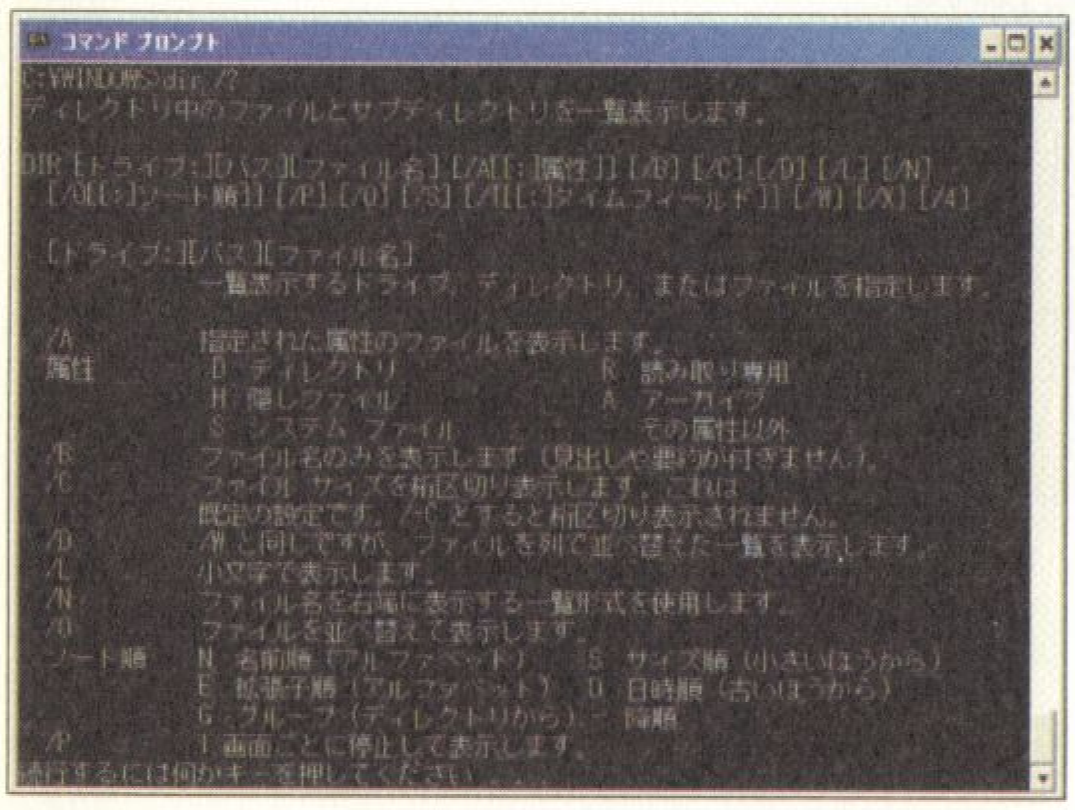
つまり、「/p」オプションを用意していないコマンドでも、スクロールさせなくてすむわけです。





コマンドプロンプトで使うコマンドの多くは、自分自身の解説を見ることができるオプションスイッチを用意しています。それは、「/?」オプションです。

例えば、「dir」コマンドで「/?」オプションを付けると、その解説を見ることができます。3の「dir」のところを**dir /?**と入れてみましょう。



ちなみに、オプションは複数指定することも可能です。その場合は、それらのオプションの効果を合わせたものになります。

gcc

人が理解できるソースコードを、コンピュータが理解できるように変換するためには、C言語の**コンパイラ**というものが必要です。Windowsで動く、C言語のコンパイラには、様々なものがあるのですが、本書では、GCC（GNU Compiler Collection）という、UNIX系システムで、最もメジャーなコンパイラを使います。

Windows用のgccは2つの種類ありますが、ここでは「Mingw gcc」を導入します。

[Cygwin]	CygwinとはWindows上にUnixライクな環境を構築するもので、その中にgccも含まれています。Unix 系のOSに慣れた人には定番のツールとなっています。
[Mingw gcc]	Mingw gccもWindows用のgccですが、Cygwin（cygwin.dll）を必要としないのが特徴で、サイズも小さくインストールも簡単です。

▶専門的な説明

●gccの日本語文字コードについて

gccは、日本語（シフトJISコード）に完全には対応していません。そのために、日本語（シフトJISコード）に基づいた開発環境では、特定の文字で問題を起こします。ここでは、本書で考えている開発環境以外の場合も考慮して、ソースコードに日本語（漢字）を使いませんでした。

※**注意**　このサイトからダウンロードできないときは、GCCのサイトにアクセスしてください。
GCCの公式サイト：http://gcc.gnu.org/、GCCのダウンロードサイト：http://gcc.gnu.org/mirrors.html

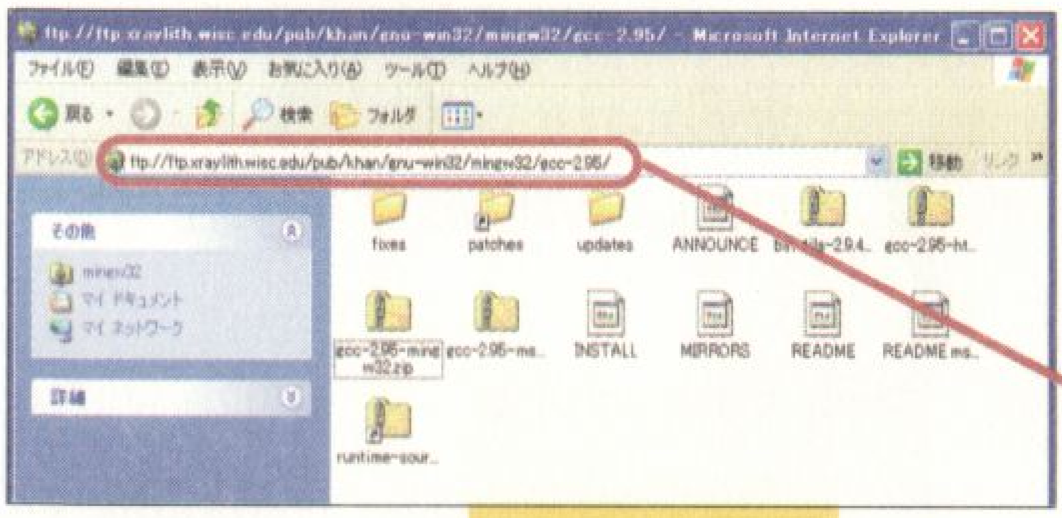
インストール

本書では、次のプログラミング環境を想定しています。

・パーソナルコンピュータは、Windowsが動作するマシン
・OSは、Windows XP

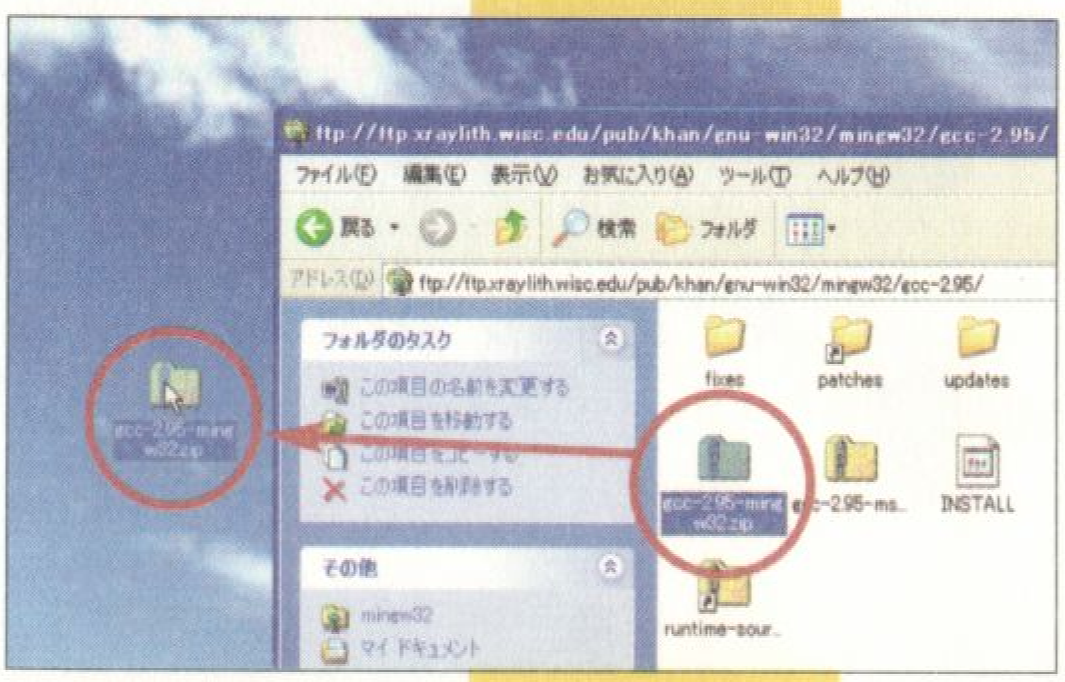
gccを、次の場所から、自分のコンピュータの適当な場所にダウンロード※してください。

ファイル名：gcc-2.95-mingw32.zip (6.8MB)
ftp://ftp.xraylith.wisc.edu/pub/khan/gnu-win32/mingw32/gcc-2.95/

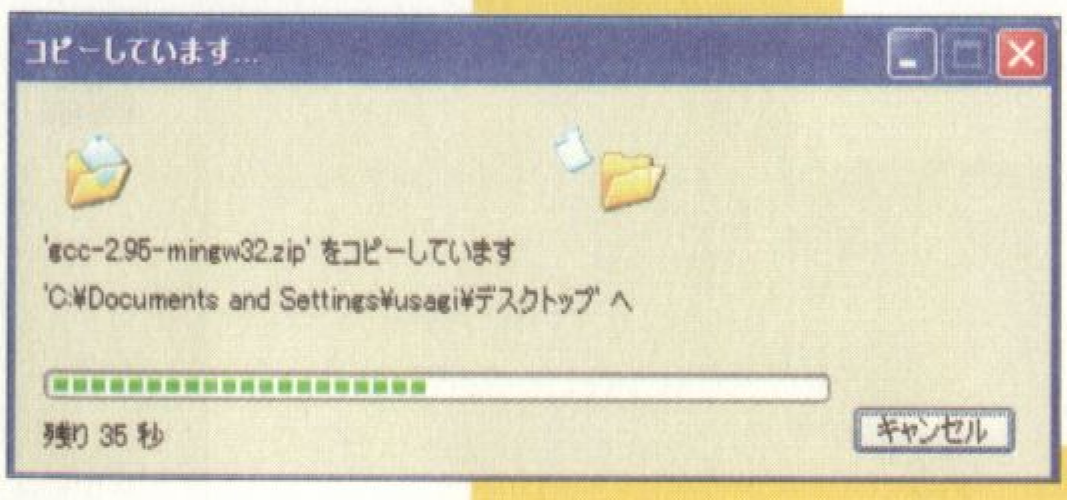


1 Internet Explorerのアドレス欄に、上記アドレスを入力します。

1 アドレスを入力する。



2 **gcc-2.95-mingw32.zip**を適当な場所に（ここではデスクトップ）にドラッグ&ドロップします。

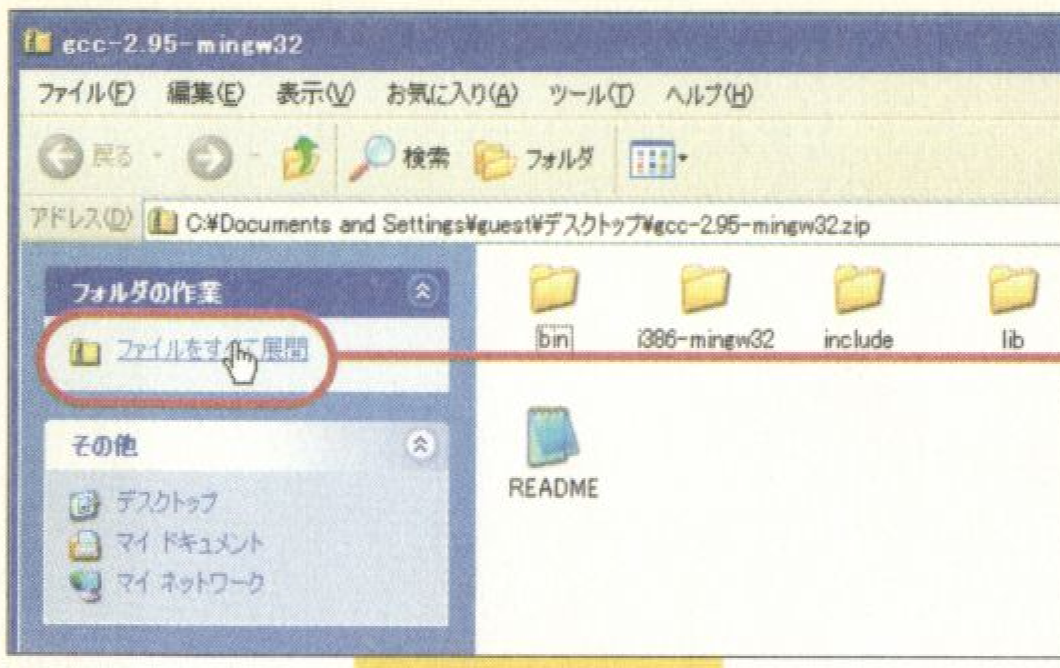


3 コピーが開始されます（コピー時間は通信環境によって長短があります）。

4 ダウンロードしたファイルをダブルクリックします。

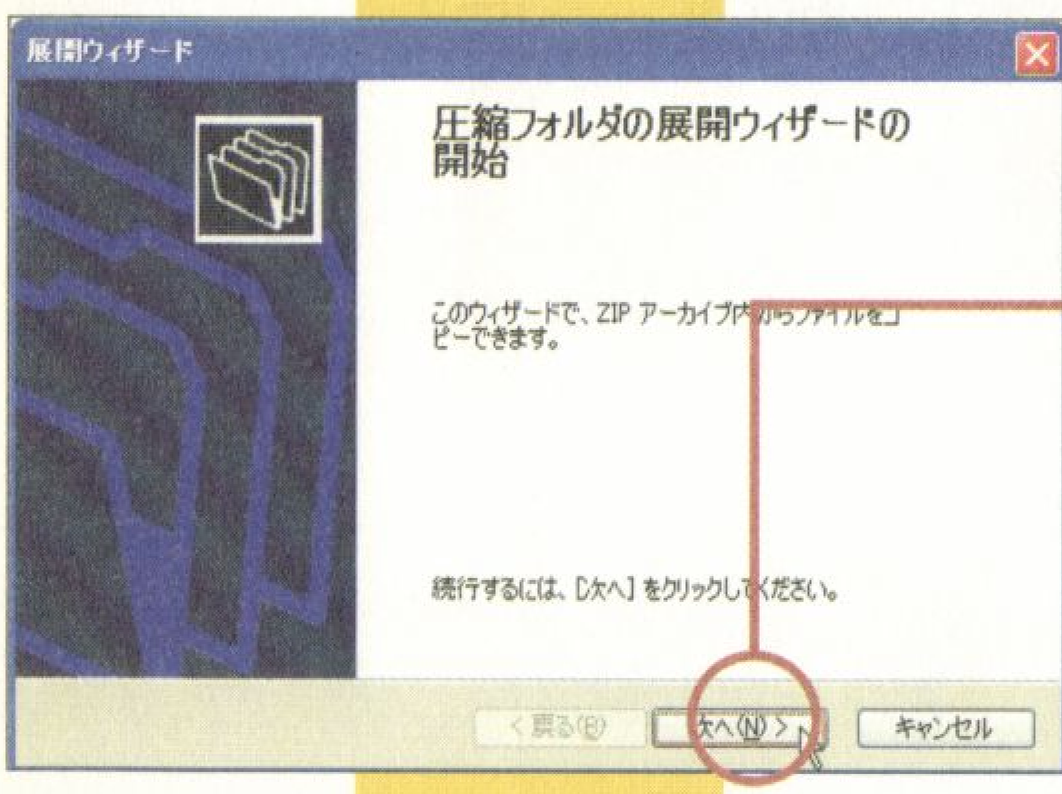


4 アイコンをダブルクリックする。



5 開いたフォルダから（**ファイルをすべて展開**）をクリックします。

5 ファイルをすべて展開をクリックする。

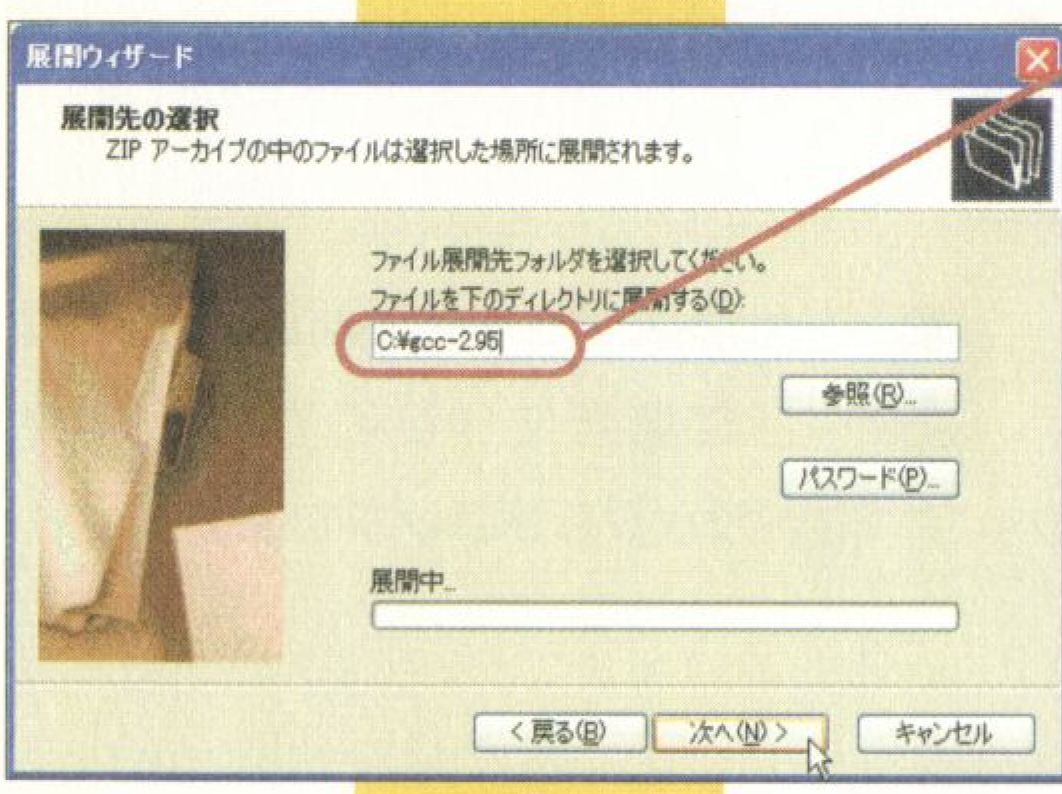


6 **展開ウィザード**で次へをクリックします。

6 次へをクリックする。

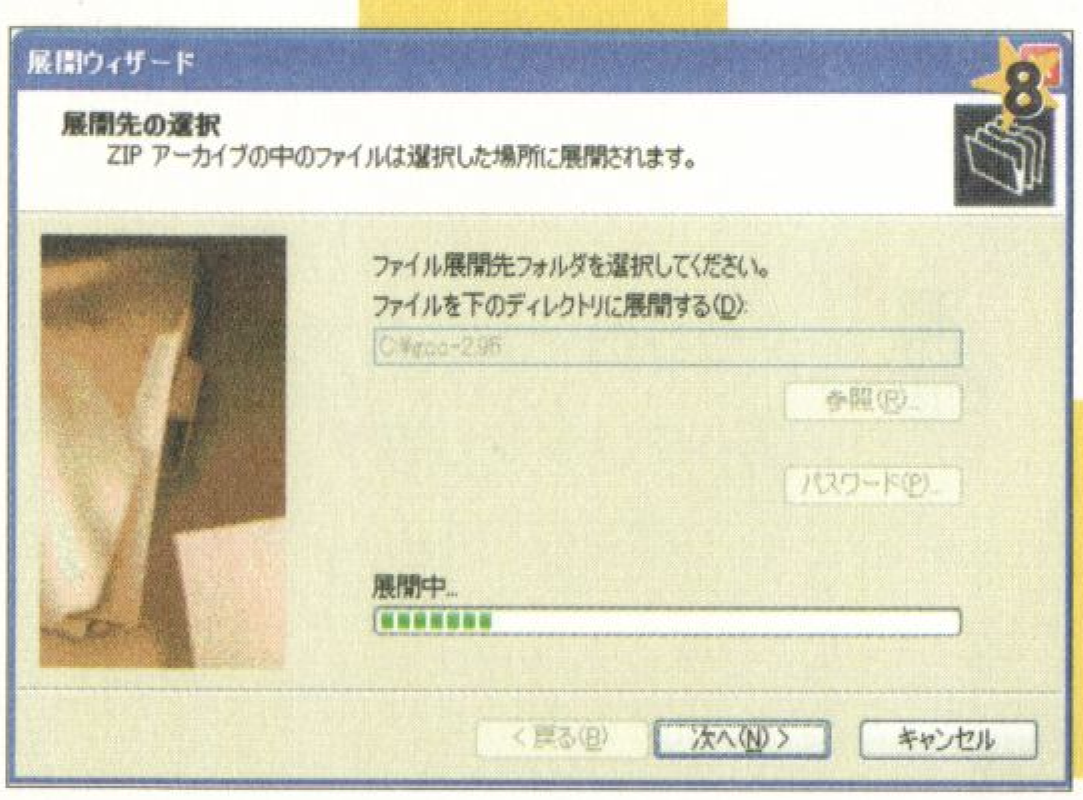
7 展開先ディレクトリに**C:¥gcc-2.95**と入力します。

7 フォルダ名を入力する。

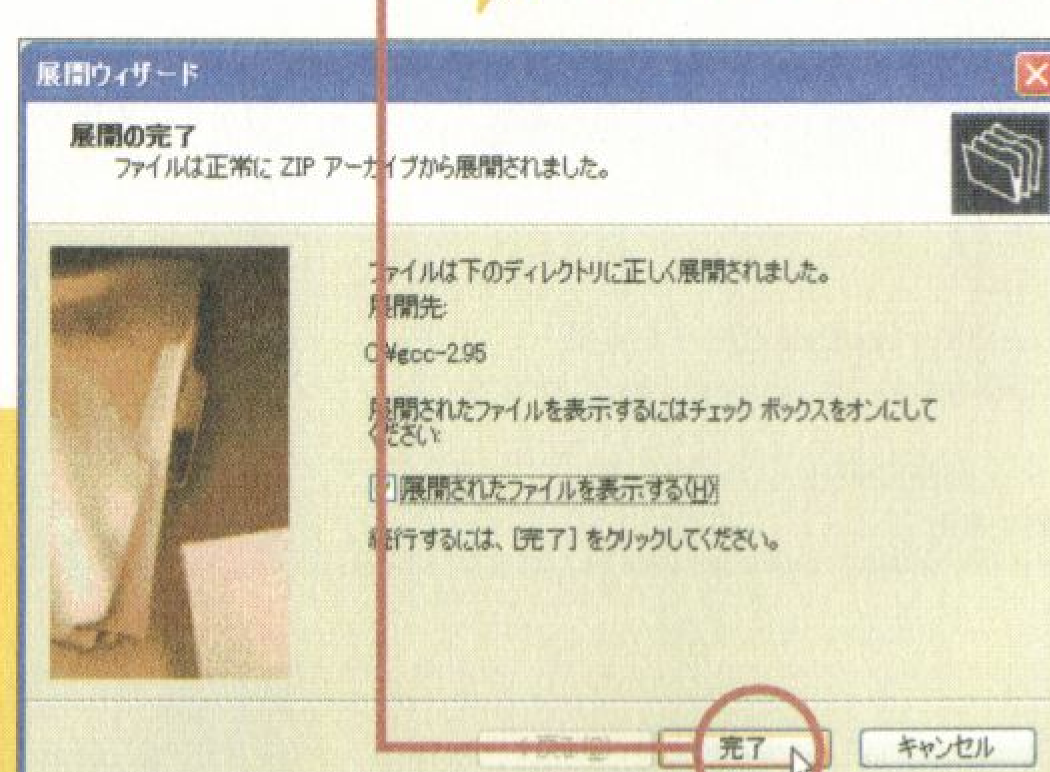


8 展開が開始されます。

9 完了をクリックします。



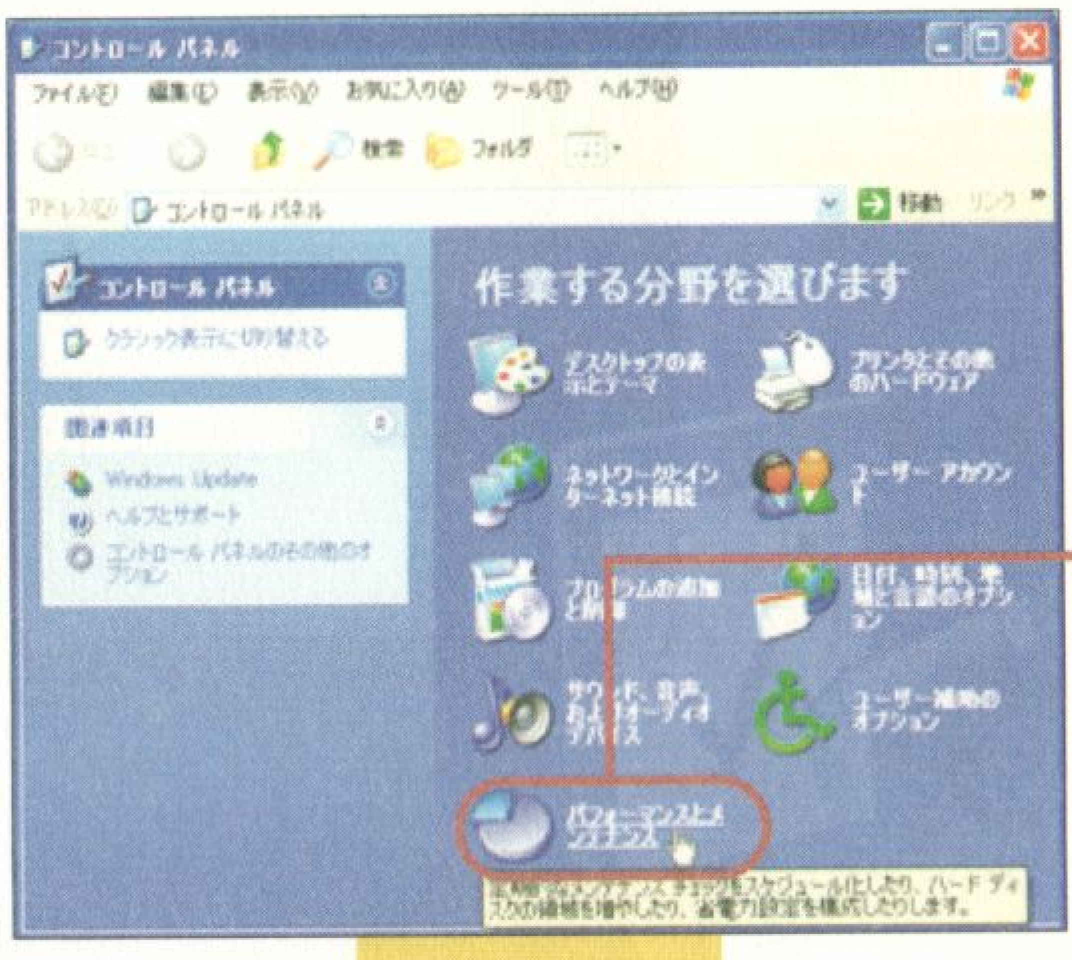
9 完了をクリックする。



※この操作はWindowsXPの場合です。Windows 2000以前のOSは解凍ソフトを別途用意してください。

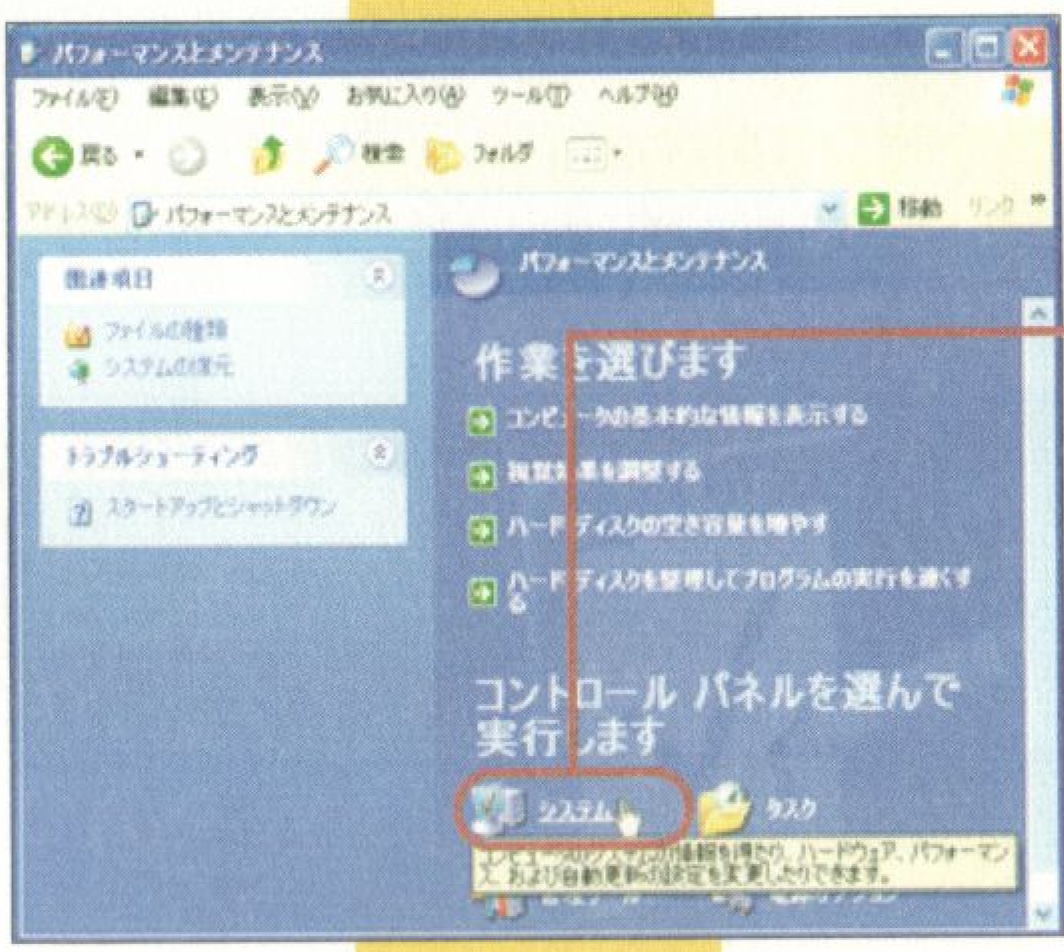
▶▶gccを使うための環境設定（パスの設定）

パスの設定とは、インストールしたgccが、どこにあるのかをコンピュータに教えてあげるための作業です。



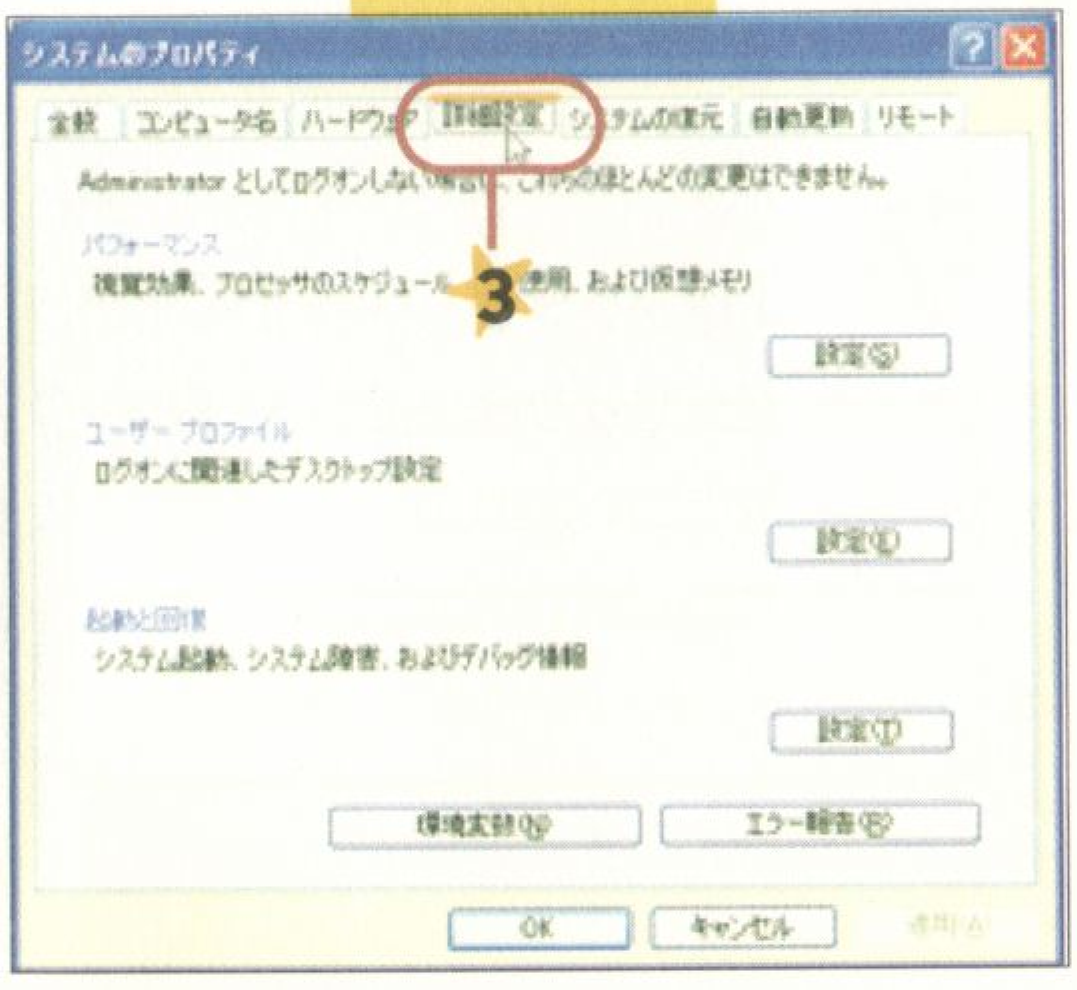
1 **コントロールパネル**を開き、**パフォーマンスとメンテナンス**をクリックします。

1 パフォーマンスとメンテナンスをクリックする。



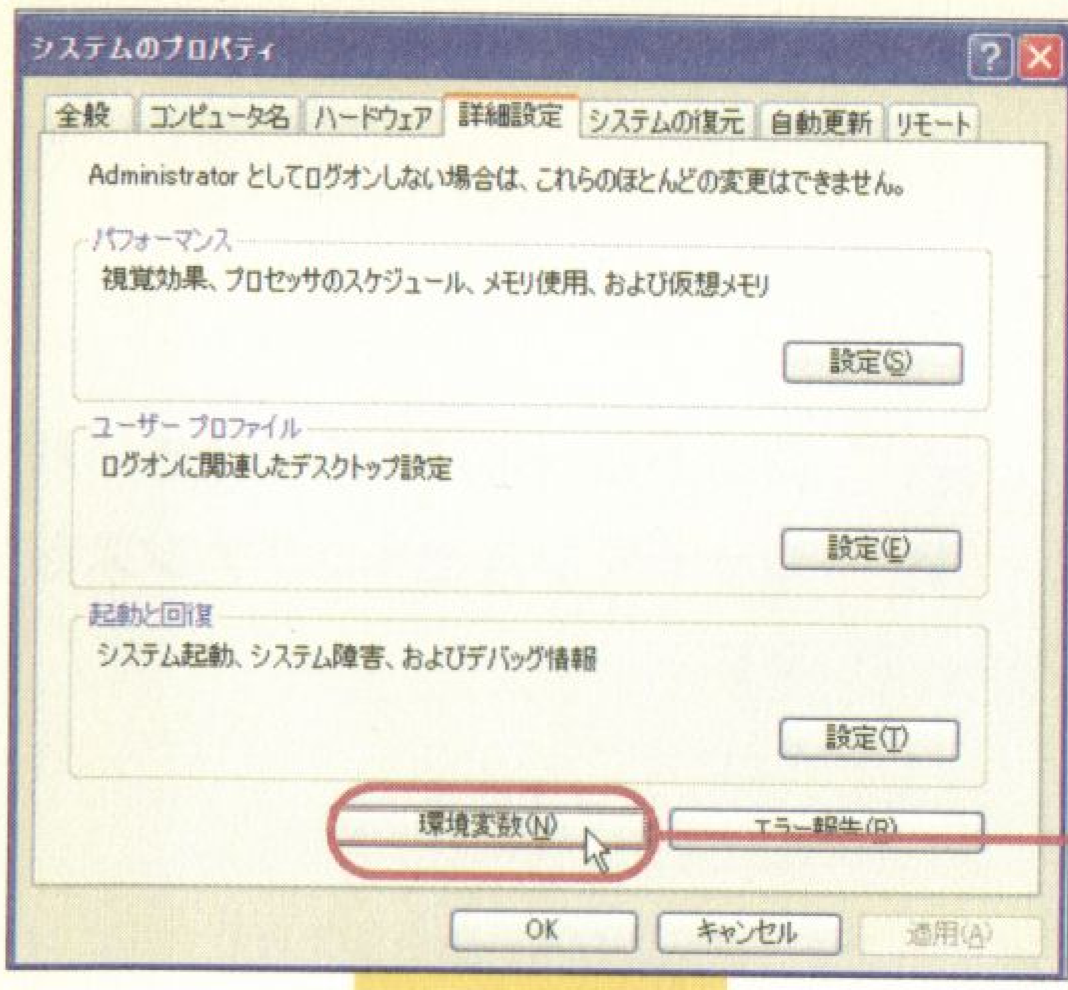
2 **システム**をクリックします。

2 システムをクリックする。



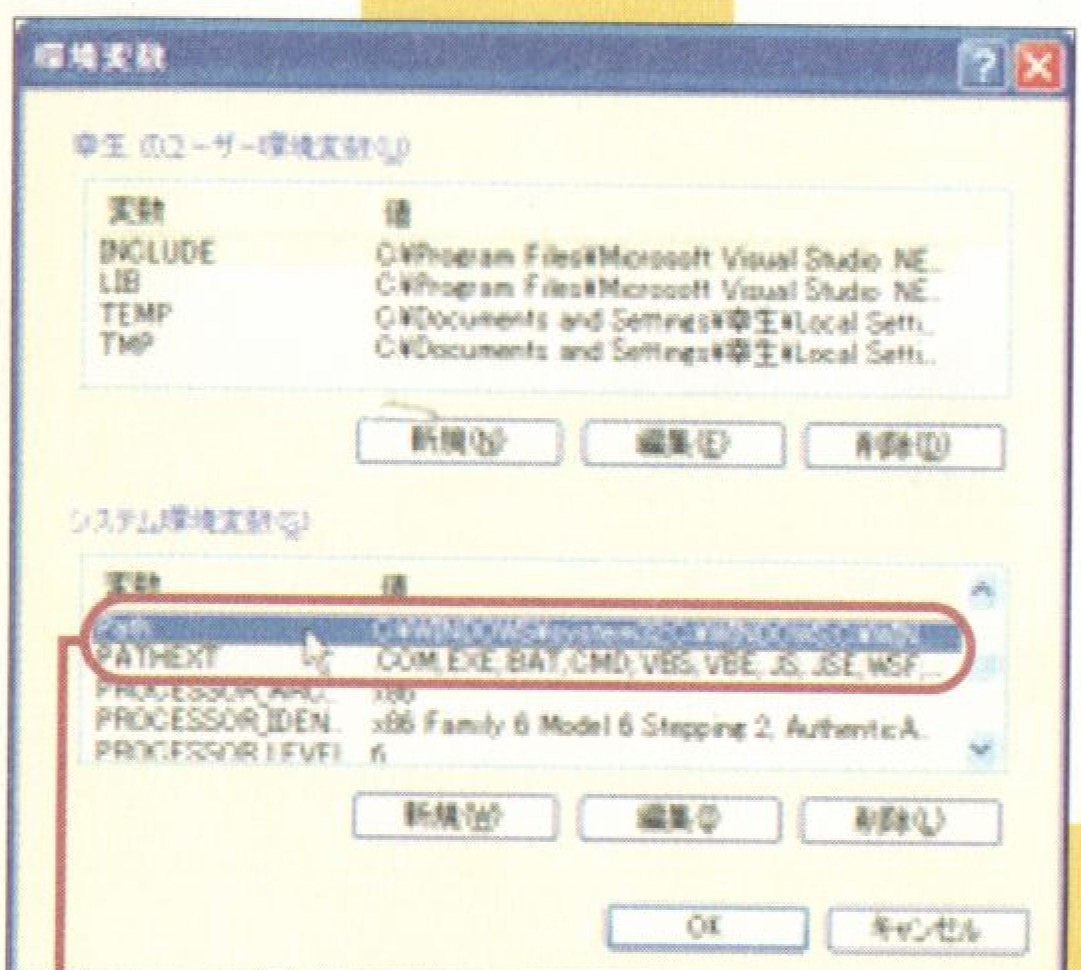
3 **詳細設定**をクリックします。





4 **環境変数**をクリックします。

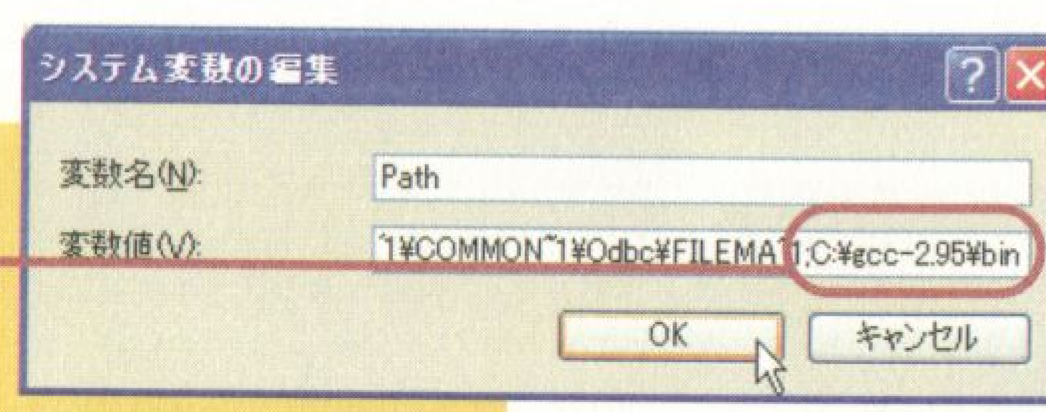
4 環境変数をクリックする。



5 Pathをクリックする。

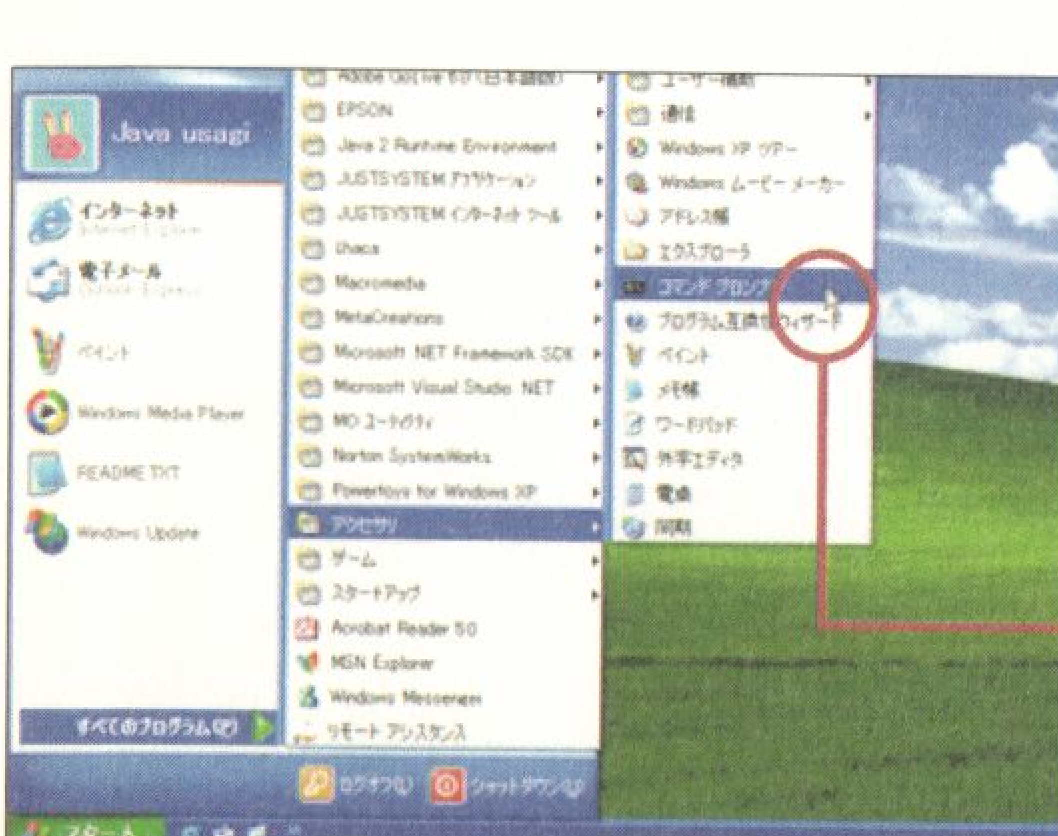
5 **システム環境変数(S)**から**Path**をクリックします。

6 変数値の最後に、「**;C:¥gcc-2.95¥bin**」を追加して、**OK**をクリックします。

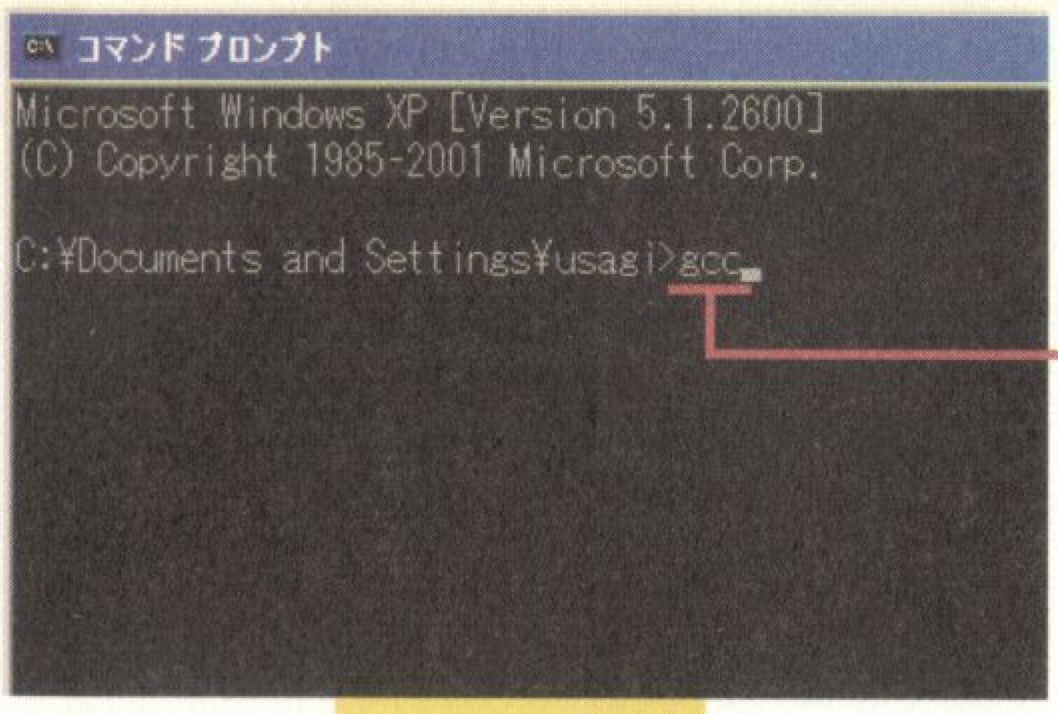


6 「;C:¥gcc-2.95¥bin」と入力する。

7 コマンドプロンプトを起動します。**スタートメニュー➡すべてのプログラム➡アクセサリ➡コマンドプロンプト**をクリックしてください。

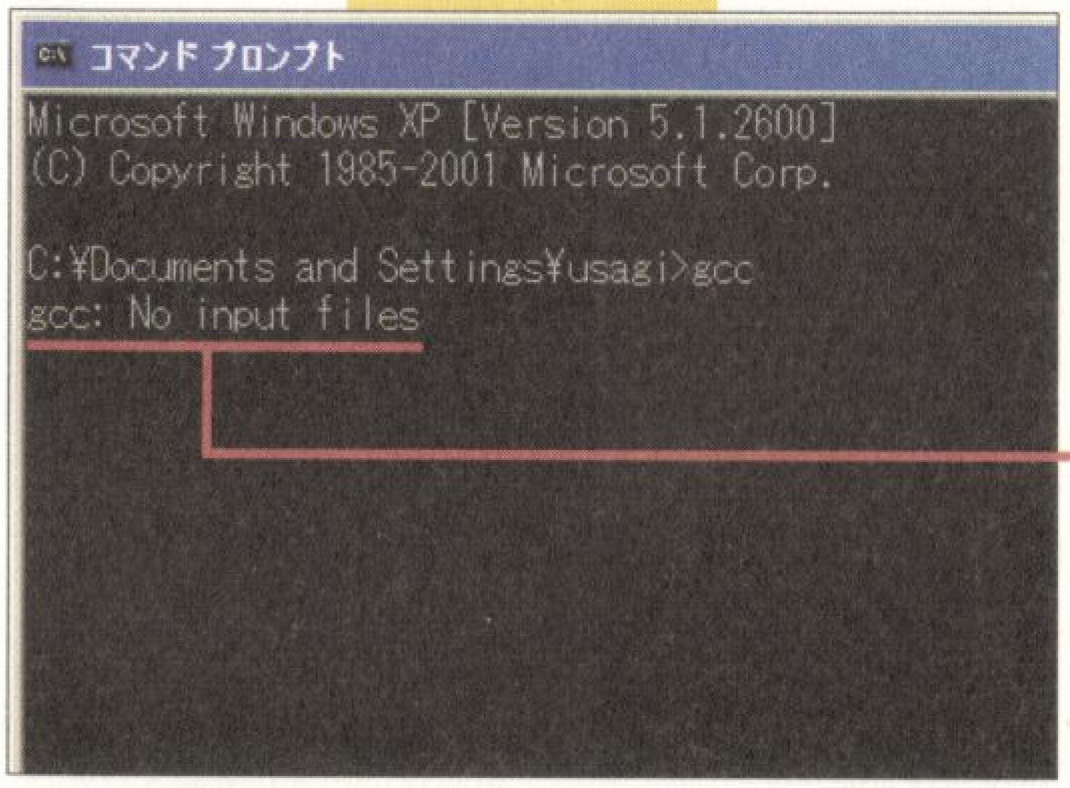


7 コマンドプロンプトをクリックする。



8 **コマンドプロンプト**に、「**gcc** Enter 」と入力します。

8 「gcc Enter 」と入力する。



9 「**gcc： No input files**」と表示されます。

9 「gcc：No input files」と表示される。



もし、表示されない場合は、パスの設定がうまくできていないことが考えられます。はじめから確認してください。

COLUMN C言語とC++言語の違い

C++言語（日本では略して「シープラプラ」とも読みます）は、C言語を基本としてこれを大幅に拡張した別の言語です。 C言語を基本としているので、C言語で記述されたプログラムをC++言語として取り扱うことはほぼ可能ですが、C++言語は、C言語に比べて抽象化が進んでいるので、逆はできません。

C++言語では、C言語にはなかった便利な文法がいろいろあります。また、現在のプログラミングスタイルの主流であるオブジェクト指向の考え方に合ったプログラムを構築するための、様々な概念や機構が備わっています。

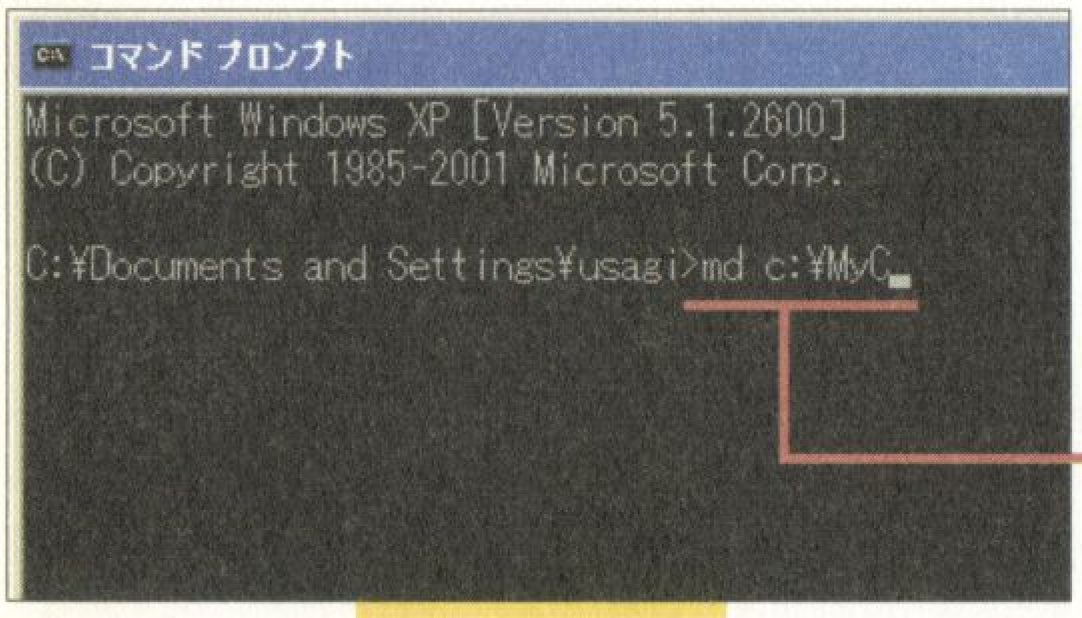
Section 02 プログラミングしてみる

本格的なプログラミングを学ぶ前に、とにかく動くものを作ってみたいと思います。ここで大切なことは、C言語を使ったプログラムの作成手順を肌で感じることです。

プログラミング用フォルダの作成

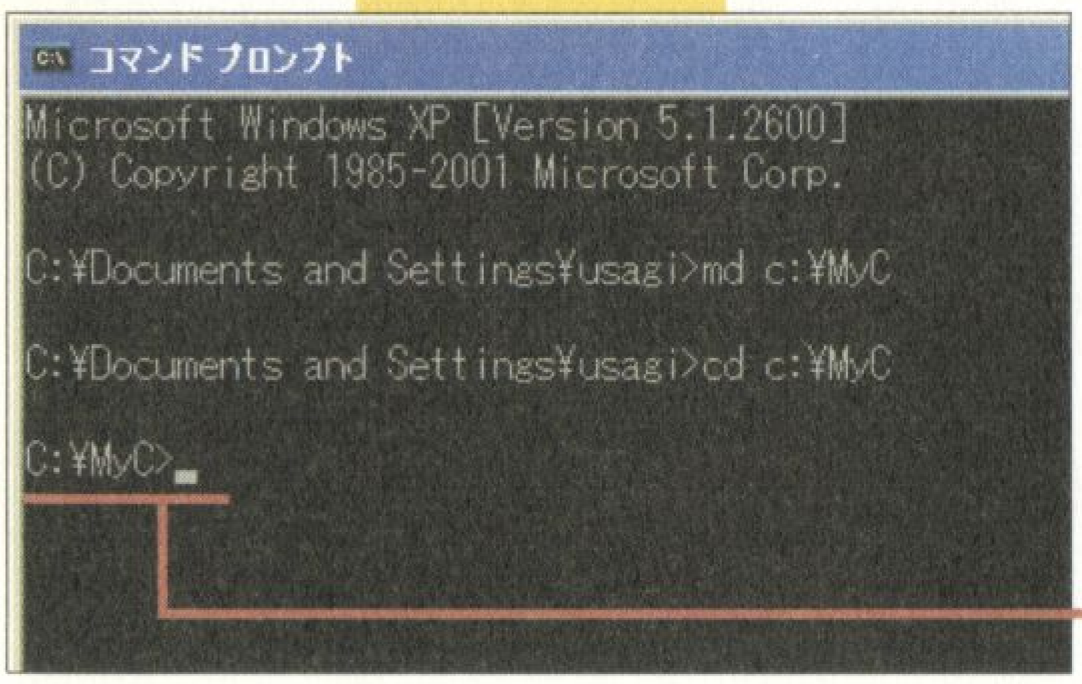
はじめに、プログラミングをするときに使うためのフォルダ「MyC」を作成しておきましょう。このフォルダに、これから作成するサンプルソースコードを保存します。

それでは、Cドライブのフォルダの中に「MyC」フォルダを作成します。



1 「md c:¥MyC Enter 」と入力する。

1 **コマンドプロンプト**を起動して、「**md c:¥MyC** Enter 」と入力します。



2 このように表示される。

2 「**cd c:¥MyC** Enter 」と入力して、図のようにフォルダが移動したら成功です。

注意!



エラーが表示された場合は、**スタートメニュー➡マイコンピュータ➡ハードディスクドライブ➡(C:)**を開き、そこに「**MyC**」フォルダがあるかどうかを確認してください。

ソースコードの入力

さて、これからC言語でプログラムを作成していきます。プログラムを作るためには、最初に**ソースコード**というものを作成します。

これは、コンピュータに指示するための「指示書」のようなもので、C言語のプログラムを作る規則に則って記述する「文」の集まりです。

ソースコードを作成していくことを、**コーディング**といいます。



ソースコード作成の注意

- 半角全角、大文字小文字の入力に注意する。
  原則的にソースコードの入力は半角文字（英数字）を使う。
- 大文字と小文字は違うものとして区別される。
  ➡例：「MY」、「my」、「My」
- 空白は、半角スペースやタブキー（Tab）を使う。
- 「行番号：(例1)：」は入力する必要はない。
- 入力し終えたら、最後に Enter で改行する。

それでは、テキストエディタ（メモ帳）を起動して、次のソースコードを入力してください。このソースコードが、何をするのか気になるかもしれませんが、あとの章できちんと説明します。

```
1:/* MyC.c */
2:#include <stdio.h>
3: この行が、空白なのは、ソースコードの読みやすさのためです
4:int main(void) { ──►「{」は7行目と対
5:インデント printf("My C Program¥n");
6:インデント return 0;
7:} ──►「}」は4行目と対
```

▶**専門的な説明**

3行目が空白なのは、ソースコードをあとから読みやすくするためで、空白にしてもしなくてもかまいません。ソースコードを書くとき、区切りとして考えられるところは、行を空白にして分けて書いたり、インデントしたほうが、ソースコードが読みやすくなります

(日本語でも、同じような規則がありますね)。

4行目を見てください。「int」と「main」の間には、1文字ぶんの空白があります。このように、意味のある(独立した)単語と単語の間は、必ず空白を入れてください。

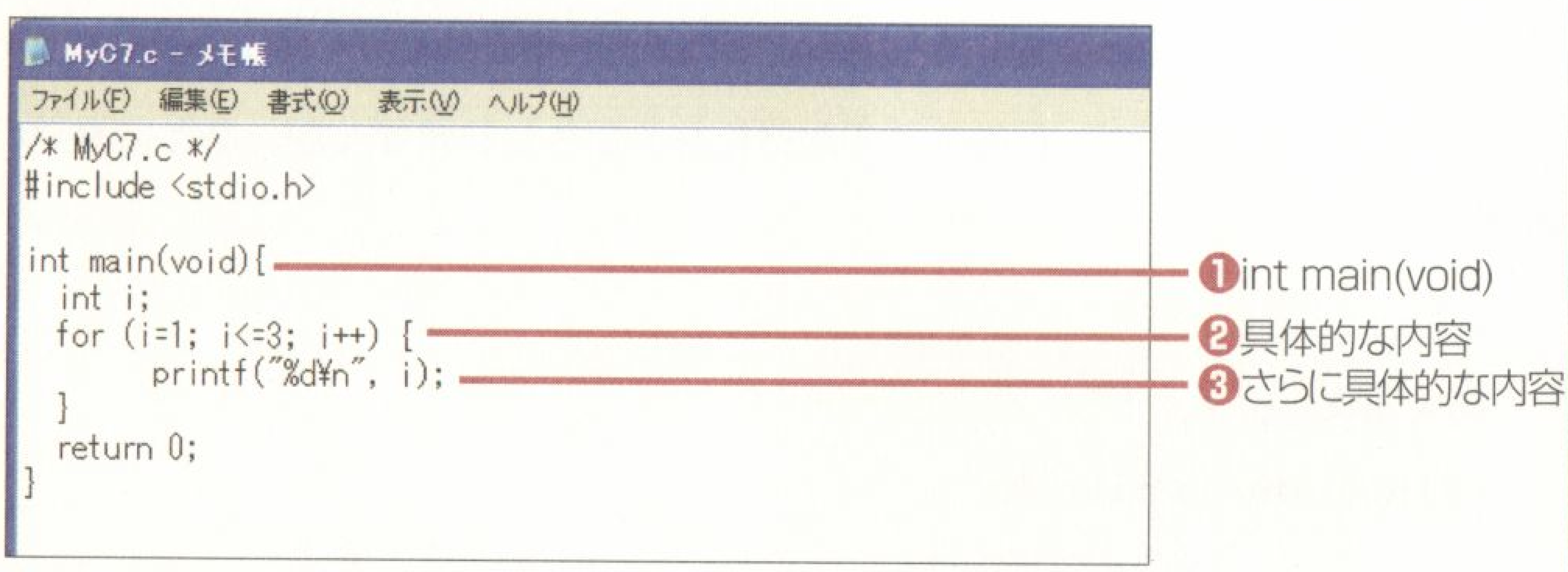
また、5行目から6行目の「インデント」というのも、ソースコードを読みやすくするために、字下げ(行の書きはじめを右にずらすこと)することです。実際に「インデント」と書くのではありません。

COLUMN インデント

インデントは、コンピュータに処理させたいことの内容が、具体的になればなるほど、ソースコードの書く位置を、右にずらすことです。

例えば、下の画面で「int main(void)」❶に比べて、「for (i=1; i<=3; i++)」❷は、より具体的ですし、「for (i=1; i<=3; i++)」に比べて、「printf("%d¥n", i);」❸は、より具体的なのです(処理そのもの)。こんなときにインデントの長さ(深さ)を調節します。

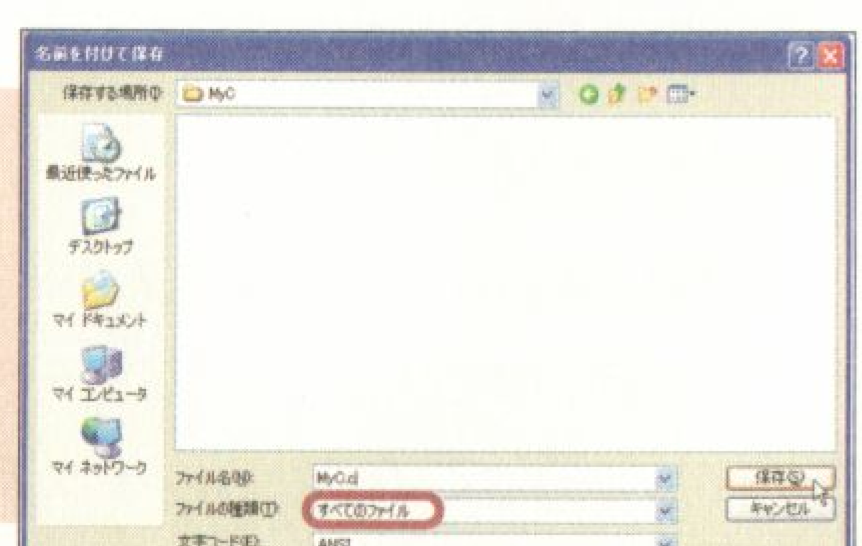
なお、インデントの長さ(ずらす文字数)は、たいてい2~8文字程度です。



上記のように入力が終わったら、**ファイル**メニューの**上書き保存**をクリックして、プログラミング用フォルダ「**MyC**」に、「**MyC.c**」というファイル名で保存します。保存したら、メモ帳を終了させましょう。



保存のとき、「ファイルの種類」を「すべてのファイル」にしてください。
また、入力のとき、大文字と小文字は区別されて違うものとなりますので、注意してください。



C言語のソースコードに名前を付けるには規則があります。

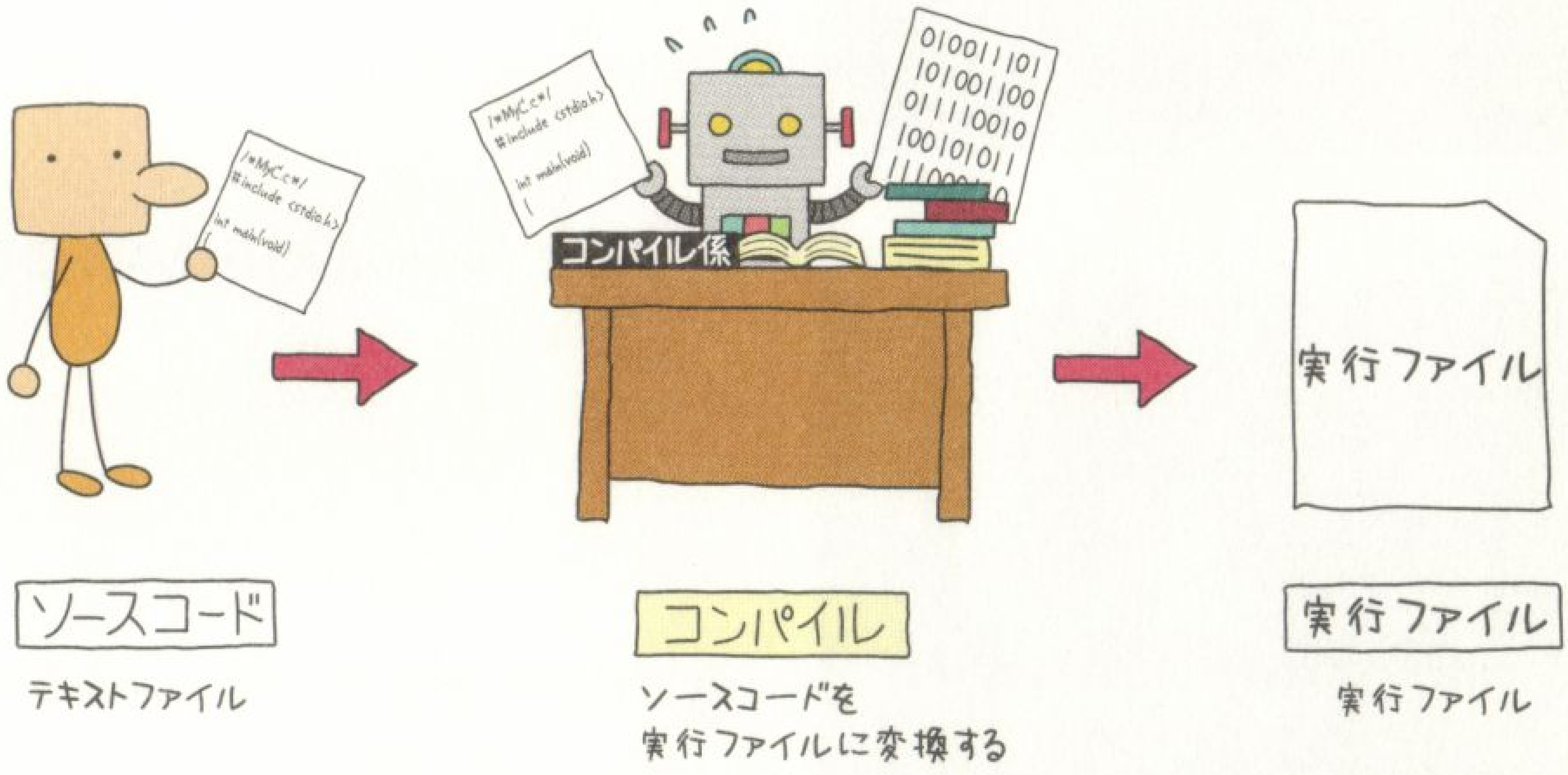
ソースコードの名前には、「**プログラム名.c**」と付けることになっています。
いま保存した、ソースコードの入ったファイルを**ソースファイル**といいます。

コンパイルする

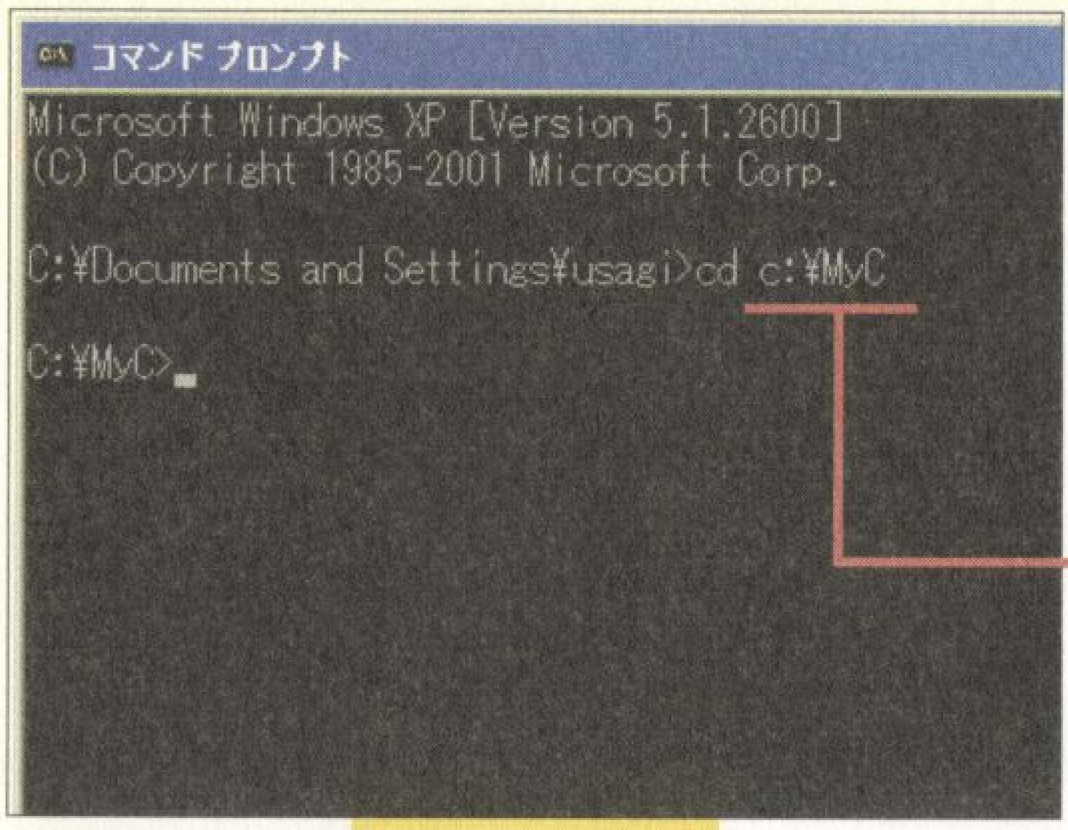
作成したソースコードは、そのままではコンピュータが理解できないので、プログラムとして動かすことができません。ソースコードをコンピュータが理解できるようにするため、機械的に変換する**コンパイル**という作業が必要です。

コンパイルの結果、人が理解できるソースコードは最終的にコンピュータの理解できる**実行ファイル**（詳しくは第7章で説明します）というものになります。

ここで、もう一度、復習のため「本文23ページ インタプリタとコンパイラの図」を参照してください。

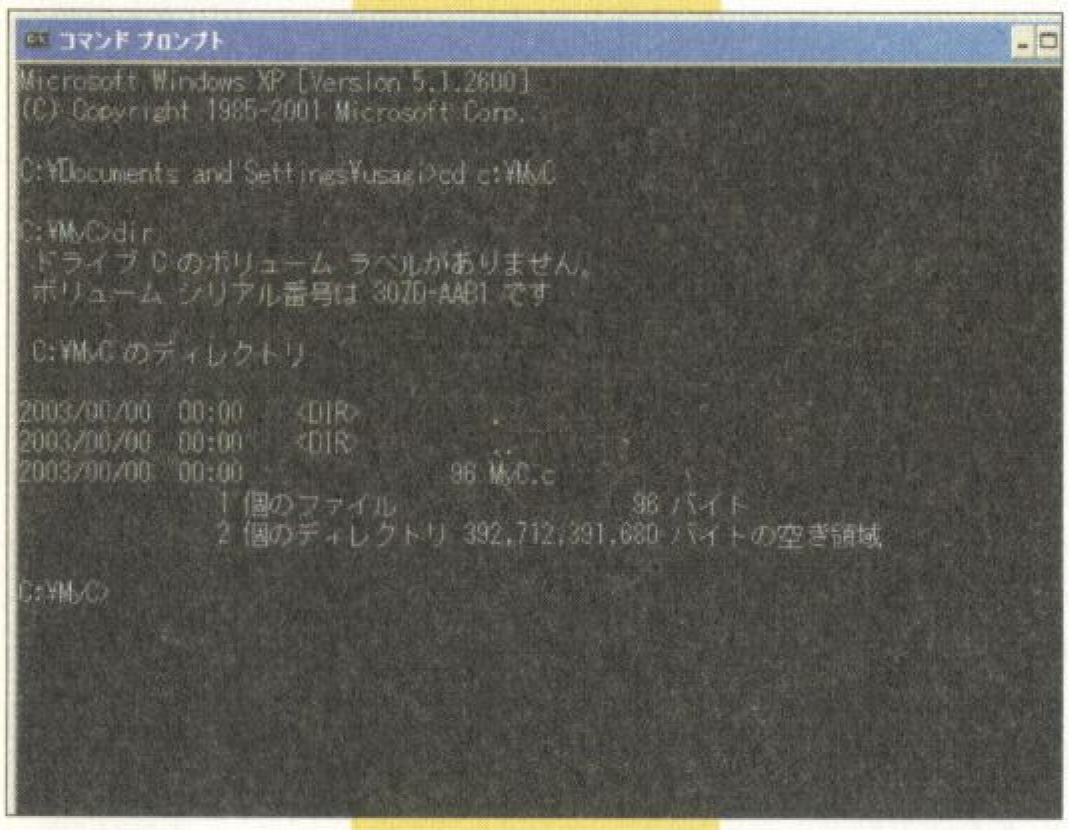


では、実際にコンパイルという作業をしてみましょう。

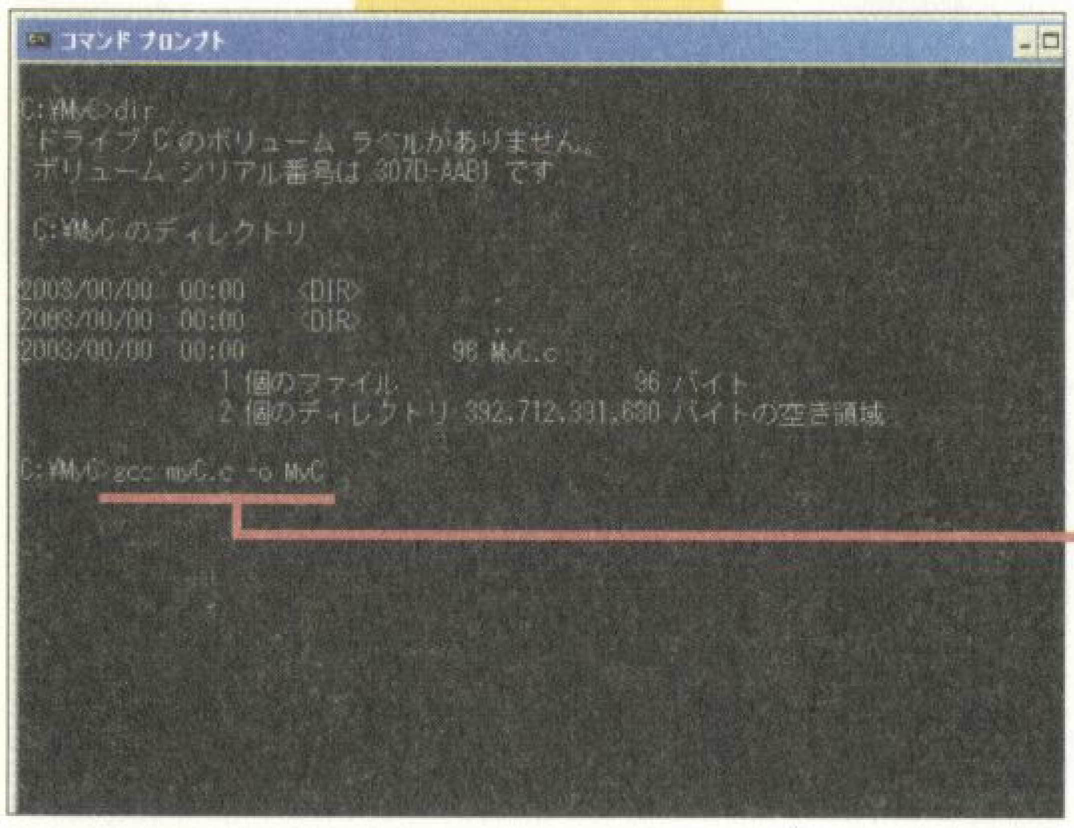


1 「cd c:¥MyC Enter 」と入力する。

1 **コマンドプロンプト**を起動して、「**cd c:¥MyC** Enter 」と入力します（プログラミング用フォルダに移動します）。



2 コンパイルをする前に、先ほど作成した「**MyC.c**」が、「**MyC**」フォルダにきちんと作成されたか、確認しておきます。「**dir** Enter 」と入力します。画面は、次のように表示されているはずです。



3 「gcc MyC.c -o MyC Enter 」と入力する。

3 「**gcc myC.c -o My C** Enter 」と入力します。

「MyC.c」をコンパイルすると、「**MyC.exe**」というファイル（実行ファイル）が作成されます。

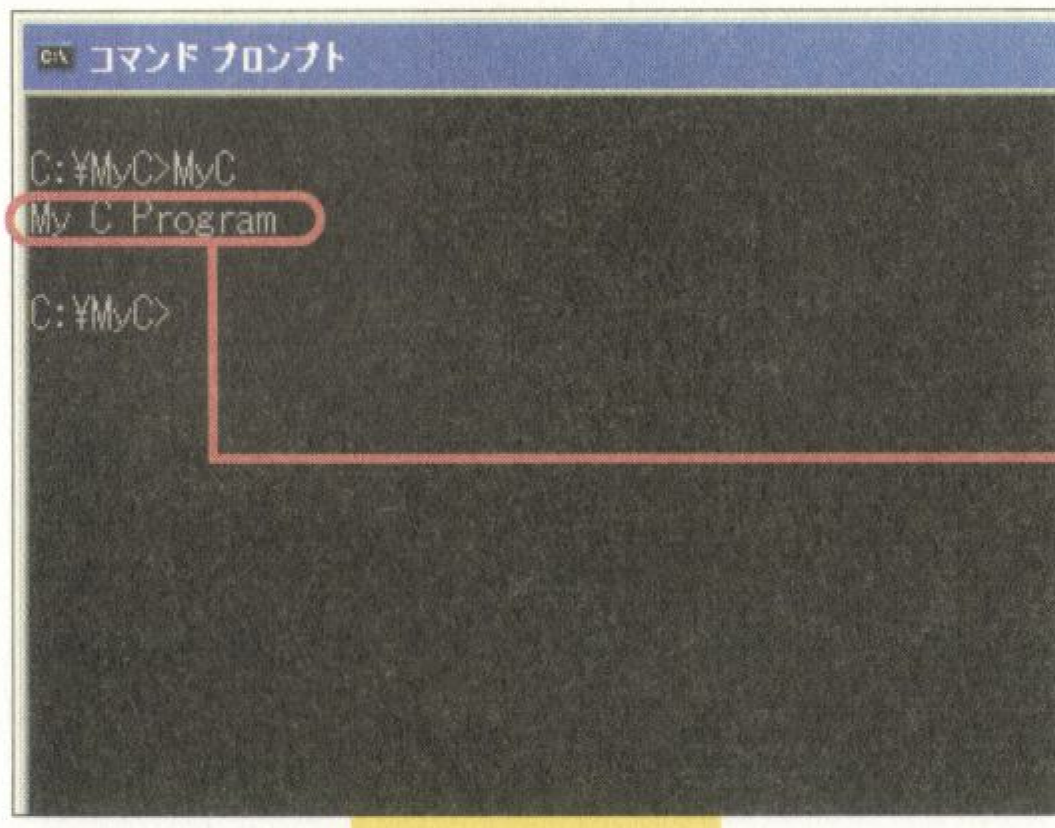
注意!



コンパイルをしたとき、エラーが表示される場合もあるでしょう。そういうときは、入力したソースコードに間違いがないかどうかを確認してください。

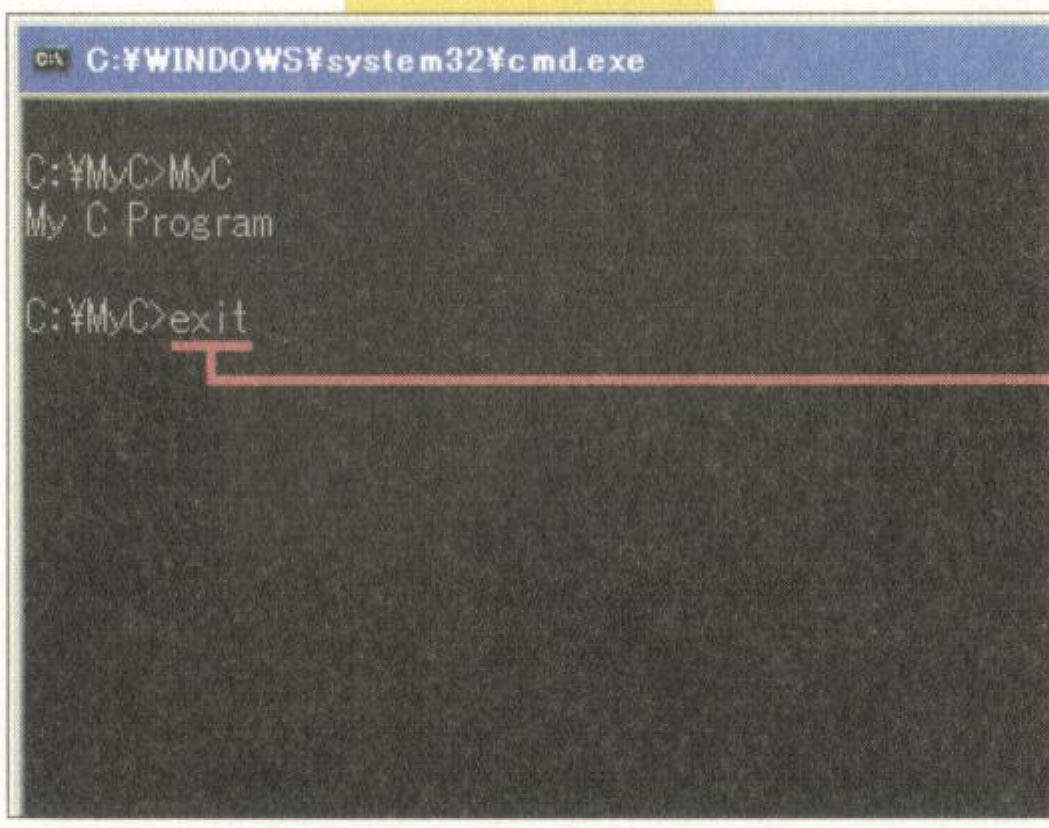
▶▶実行する

次は、コンパイルしたファイルを実行してみましょう。



1 「**MyC** Enter」と入力します。画面に「**My C Program**」と表示されます。

1 「My C Program」と表示される。



2 **コマンドプロンプト**を終了します。「**exit** Enter」と入力します。

2 「**exit** Enter」と入力する。

ここでもう一度、作成手順の確認をしておきます。

●作成手順のまとめ●

❶テキストエディタで、「プログラム名.c」を作成します。

❷gccコマンドで、ソースファイルをコンパイルして、「プログラム名.exe」を出力します。

❸プログラム名で、プログラムを実行します。

あわせて「本文23ページ インタプリタとコンパイラ」の図を参照してみてください。

▶▶思ったように実行できないときは

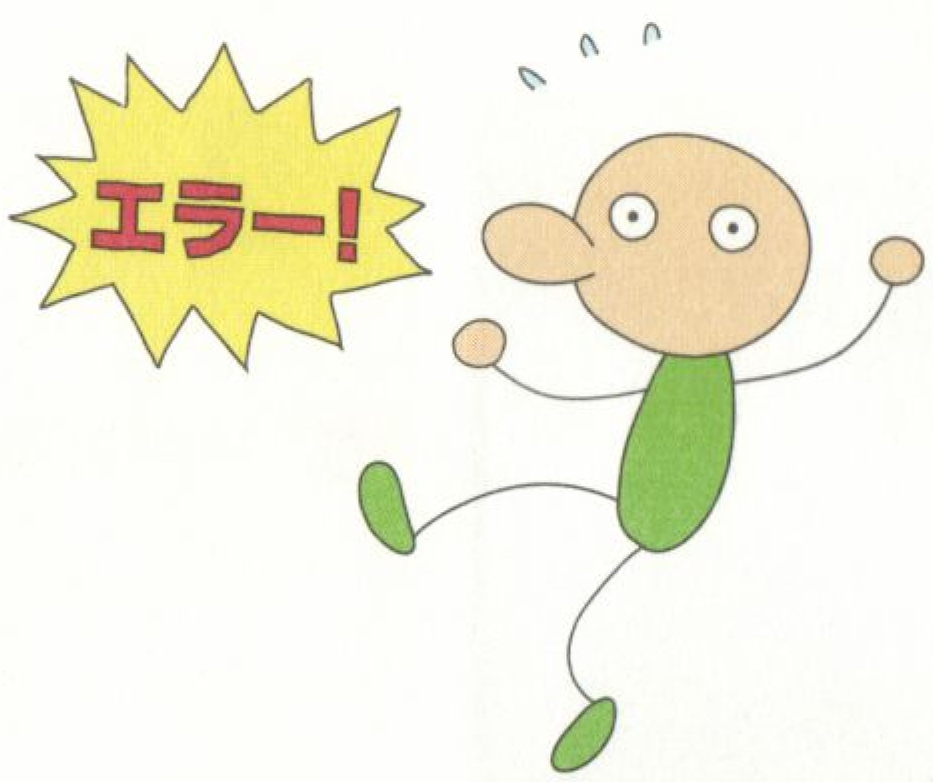
どうですか、思ったように実行できたでしょうか。

もし、ここで説明した作成手順で、ここに書いてあるとおりのソースコードを入力したにもかかわらず、思ったように実行できていないときは、手順をもう一度確認してください。それでもダメな場合は、ソースコードをもう一度確認してください。

例えば、「;」を付け忘れてはいませんか？

コンピュータは融通がきかないので、たった1つの間違いも許してはくれません。

このようなコンピュータ（プログラム）にとっての不具合を、**バグ**といいます。バグには「;」を忘れるというようなコーディングの記述ミスのようなものもあれば、プログラムを作っていく最初の段階で、そのプログラムを作る考え方が間違っていた結果、発生する構造的なミスのようなものもあります。



プログラム（プログラミング）には、バグはつきものです。

なぜなら、プログラムを実行するのは完全無欠なコンピュータですが、作るのはミスの可能性がある人だからです。

そのため、バグを完全になくすことは不可能といってよいでしょう。しかし、できるだけ少なくする努力は必要です。そうでなければ、プログラムの品質、ひいては信用にかかわりますし、それが人の命に関わるような現場で使われていたならば、もしかすると大事故につながる場合もあるからです。

先ほど、バグをできるだけ少なくする努力が必要だと説明したのですが、このバグを無くしていく作業のことを、**デバッグ**といいます。

実は、プログラムの作成とは、このデバッグとテストの繰り返しのことといっていいぐらいにデバッグは重要です。

書かれたソースコードが、そのままで動作することは、少し大きめのプログラムを作りはじめたら、まれなことです。ふつうは、記述ミスや文法ミスなどによる、エラーがたくさん検出されることでしょう。

例題 バグとデバッグ

次のようなソースコードを入力して、作成手順に沿って実行してみましょう。

```
1:/* MyC.c */
2:#include <stdio.h>
3:
4:int main(void) {
5:                printf("My C Programming¥n")
6:                return 0;
7:}
```

●解答

5行目の最後に「;」がないので6行目でコンパイルを停止しましたというエラーメッセージが画面に表示されましたね。このような記述ミスによるエラー表示は、これからも目にすることと思いますが、めげずにデバッグしていきましょう。




```
MyC.c: In function`main' :
MyC.c:6: parse error before`return'
```

COLUMN エディタを変えてみよう

Windowsに標準で付属している「メモ帳」以外にも、エディタソフトには様々な種類のものがあります。インターネットで入手できるフリーウェアからパッケージソフトまで、使い勝手もいろいろです。

行番号が表示できたり、カットアンドペーストの機能が拡張されていたりと、それぞれに便利な機能があるので、自分の好みに合ったエディタを探してみるのもよいでしょう。

practice 練習問題

解答は213ページ

1 テキストエディタ　難易度★★★☆☆

テキストエディタとワープロの違いを説明してください。

▶▶ヒント：本文32ページ参照

2 コマンドライン　難易度★★☆☆☆

コマンドプロンプト上で、「type」コマンドを使って、ソースコード「MyC.c」のファイル内容を表示してみましょう。

▶▶ヒント：本文34ページ参照

3 ソースコード　難易度★☆☆☆☆

次の問いについて、○か×で答えてください。

❶ C言語のソースファイルは、そのままで実行できます。

❷ ソースコード中にインデントを挿入するときは、「インデント」と書かなければいけません。

❸ ソースコード中の、「MY」と「My」は区別されます。

▶▶ヒント：本文43ページ参照

memo

Chapter2のまとめ

★CUI

CUIとは、コンピュータを文字で操作することです。

★GUI

GUIとは、コンピュータをマウスなどのポインティングデバイスと絵を使用して、直観的に操作することです。

★C言語のプログラミングに必要なもの

C言語のプログラムをするには次の3つが必要です。

・テキストエディタ
・コマンドライン
・Cコンパイラ（gcc）

★パス

パスとは、ファイルの格納場所を表す文字列のことで、ファイルの住所のようなものです。

★ソースコードの書き方

ソースコードは次のように書きます。

・基本的に、半角英数字で入力します。
・空白は、半角スペースやタブキーを使います。大文字と小文字を区別します。

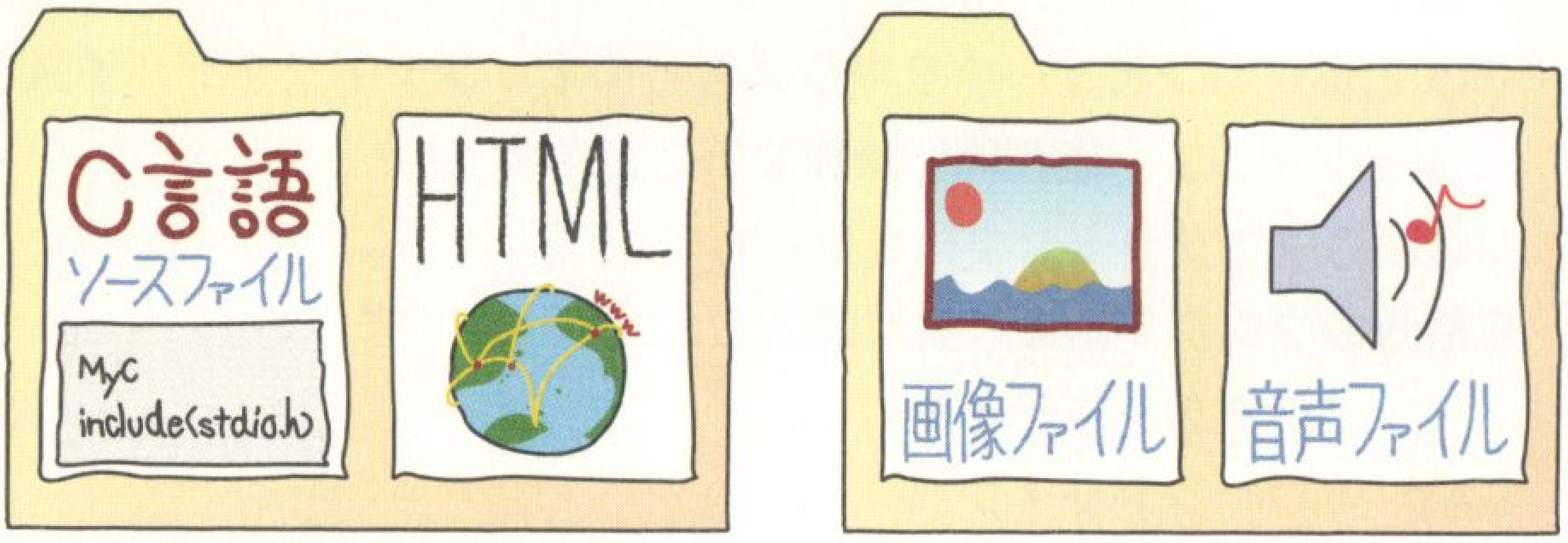
★ソースコードの作成手順

ソースコードの作成は次の手順で行います。

❶ **テキストエディタ**で、ソースファイル「**プログラム名.c**」を作成します。
❷ gccコマンドで、**ソースファイル**を**コンパイル**して、「**プログラム名.exe**」を出力します。
❸ **プログラム名**コマンドで、**プログラム**を実行します。

COLUMN

ファイルにはこんな種類がある！

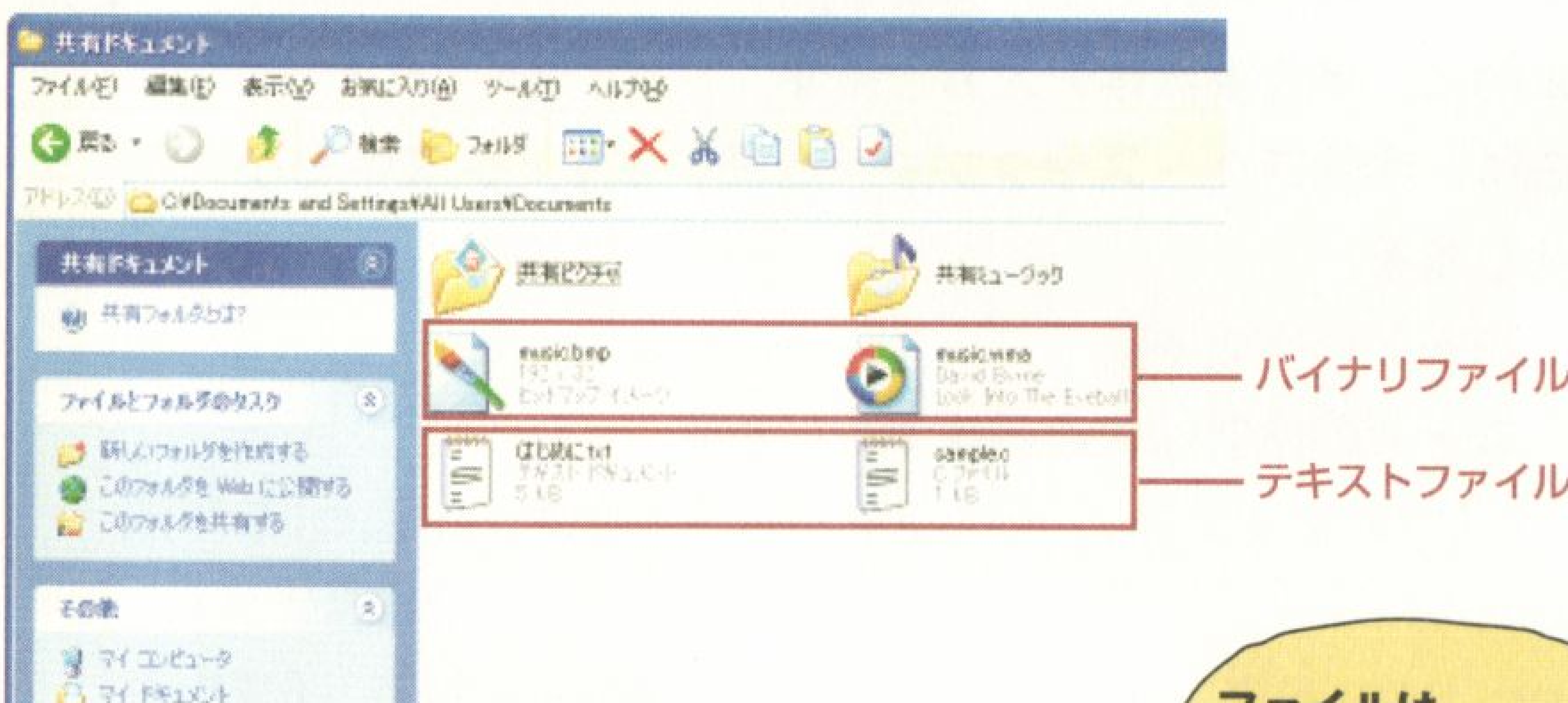


テキストファイル

文字として読めるファイルのこと
C言語ソースファイル、HTMLファイルなど

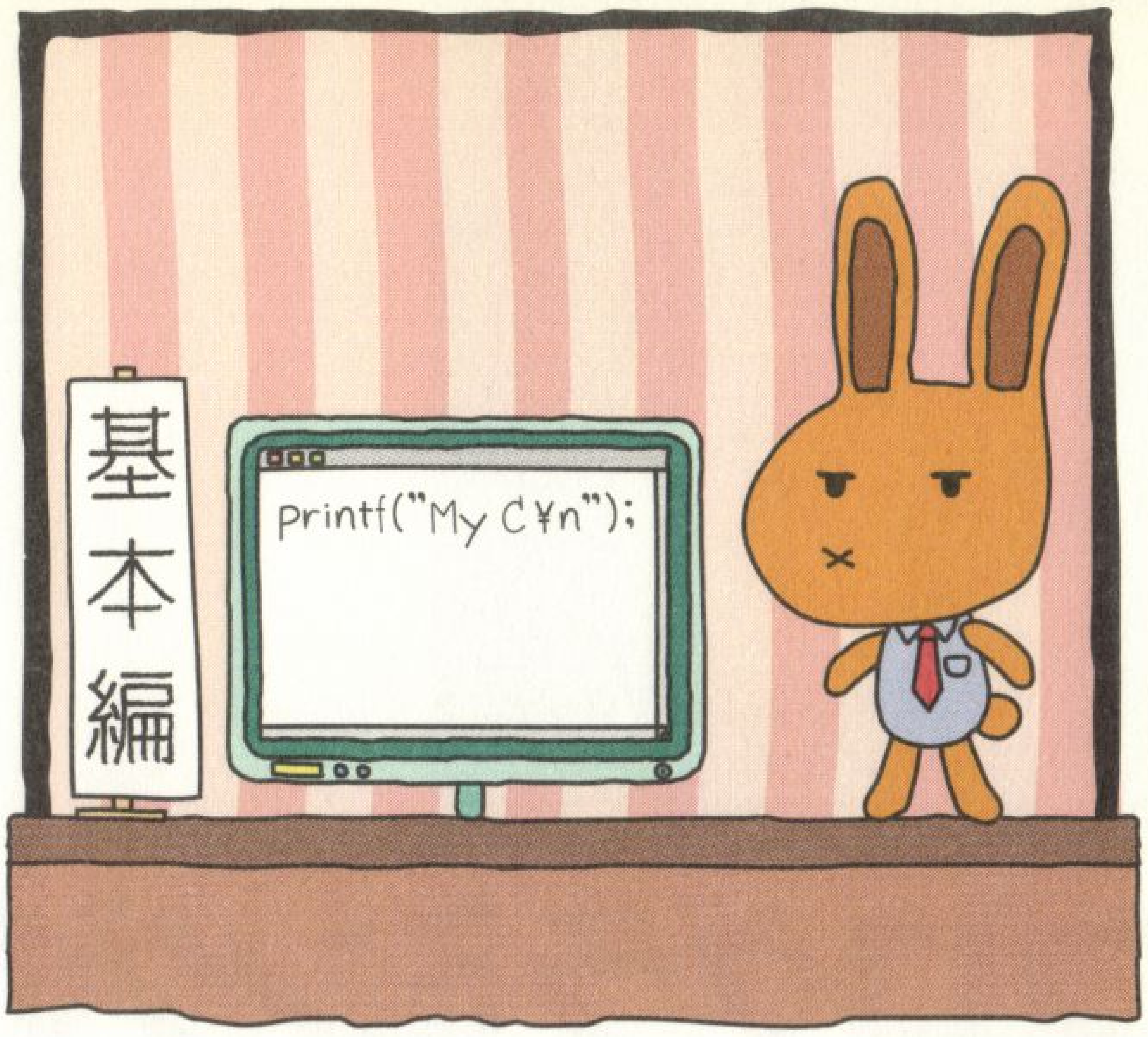
バイナリファイル

文字として解釈できないファイルのこと
画像ファイル、音声ファイルなど





ファイルは大きく分けて、**バイナリファイル**と**テキストファイル**に分けることができます。テキストファイルとは、人間が読めるような文字で書かれたファイルのことで、バイナリファイルとは、文字として解釈できないファイルのことをいいます。



★さて、準備はできましたね。

いよいよC言語のプログラミングの第一歩を踏み出します。

それではC言語の基本をしっかり押さえましょう。

★すでにC言語の基本を理解していて、

プログラミングからはじめたいという方は

➡第4章本文72ページへ

★C言語の制御文を理解したい方は

➡第5章本文94ページへ

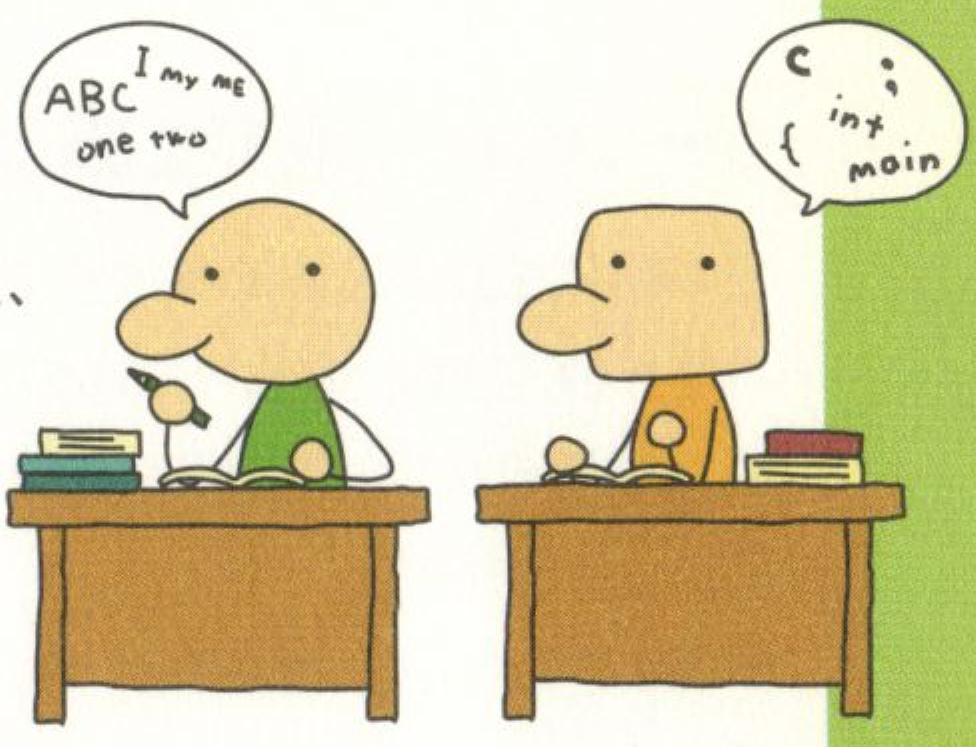
Chapter 3

プログラミングの基本



この章でマスターすること

学校ではじめて英語を習うときは、文法などの規則がありました。同じように、プログラミング言語を使ってアプリケーションを作るためにソースコードを書くときにも、守らなければならない規則があります。C言語でプログラミングをするために、C言語の文を作るための基本的な規則を知りましょう。



また、日本語の文を作るための規則を理解することと、文章を書くことが違うように、C言語の文を作るための規則を理解することと、プログラミングすることとは違います。

プログラミングは作文と同じです。つまりプログラミングとは、頭の中にある「コンピュータにこれをさせたい」というイメージを整理して、それを文にしていくことです。

それでは、プログラミングのはじめの一歩を踏み出しましょう！



C言語必須用語

関数　コメント
main関数　予約語
プリプロセッサ制御文

ステップアップ用語

アルゴリズム　解ける問題
解けない問題
実際に解けない問題

Section 01

文を書くための規則
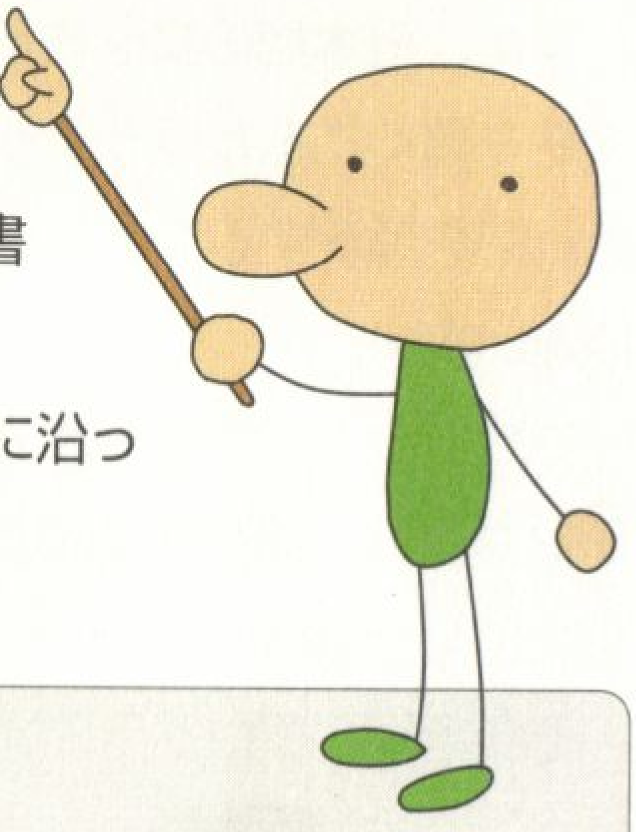
ここではC言語プログラムにおけるソースコードの基本形について、説明していきます。

ソースコードを見てみよう

日本語にもルールがあるように、プログラミング言語は、それぞれの言語の文を作るための規則を持っています。

当たり前ですが、この規則を知らなければ、ソースコードを書いていくことができません。

第2章で入力した「MyC.c」を、C言語の文を作るための規則に沿って見ていきましょう。



```
1:/* MyC.c */
2:#include <stdio.h>
3:
4:int main(void) {
5:  printf("My C Program¥n");
6:  return 0;
7:}
```

▶▶ 文

5行目を見てください。

```
5:  printf("My C Program¥n");
```

このような1行をC言語では、**文**（**単文**）といいます。

文の種類には、「プリプロセッサ制御文」、「宣言文」、「制御文」、「その他（コメント、式、ブロック、何もない文）」があります。文の種類については、その種類にあたる文が出てきたところで説明できます。

ここでは、「printf("My C Program¥n")」が出てきたので、この文について説明しましょう。「printf("My C Program¥n")」は、**式**（本文85ページ参照）です。

式には、「代入式」、「関数呼び出しの式」などがあります。「printf("My C Program¥n")」は、「関数呼び出しの式」です。

文の最後には、日本語なら「。」にあたる「;」（半角セミコロン）を必ず付け加えます。

▶▶ブロック

4行目と7行目を見てください。

```
4:int main(void) {
5:  printf("My C Program¥n");
6:  return 0;
7:}
```

ブロック
文
関数

「{」から「}」までを**ブロック**（**複文**）といいます。ブロックは、ソースコード中の区切られたひとまとまりの部分です。日本語なら、章や節、段落にあたります。

▶▶関数

4行目からはじまり、7行目で終わる、いちばん外側のブロックのことをC言語では、**関数**といいます。関数中には、コンピュータにしてもらいたいことを、文やブロックで書いていきます。日本語ならば、章にあたります。

この「printf」というのは、「" "」で囲まれた文字を、画面へ出力するように、コンピュータに指示するものだとここでは考えてください（あとの章では、また違った使い方も出てきます）。

▶▶コメント

1行目を見てください。

```
1:/* MyC.c */
```

/*（半角スラッシュとアスタリスク）から、*/までが**コメント**です。コメントというのは、ソースコードの中に書き込むメモのことです。内容は何でもかまいません。

例えば、「このプログラムは何をするのか」や「この部分は何をやっているのか」、「いつ、このソースコードを作成したのか」などを書き込みます。

▶▶main関数

4行目を見てください。「main」とありますね。

main関数は特別な関数です。C言語のソースコードには、必ずこの「main関数」が必要です。C言語のプログラムは、このmain関数から処理が始まるようになっています。main関数内の最初の文から処理を進めて、最後の文を処理し終わるとプログラムが終了するようになっています。

ここでは、main関数の「int main(void)」については、こういうふうに書くんだという、おまじないとして考えてください（第6章で詳しく説明します）。

▶専門的な説明

main関数の書いてある行を見ると、2つの単語（「int」、「void」）がありますね。

このような単語のことを**予約語**といいます。予約語とは、ソースコードを記述するときに基本となる単語をあらかじめC言語側で決めたもので、ユーザーがこの単語を勝手に使ってはいけません（付録207ページ参照）。

▶▶インクルード

2行目を見てください。

```
2:#include <stdio.h>
```

この「#」がついた部分は、「MyC.c」をコンパイルするときに、「stdio.h」というファイルをいっしょにコンパイルすることを指示しています。日本語でいうと、「脚注」のようなものです。なお、「#」がついた行のことを**プリプロセッサ制御文**といいます。「#」は、「この行は、プリプロセッサ制御文だよ」という目印になります。

「#include <stdio.h>」については、第7章で説明しますので、ここでは、「#include <stdio.h>」は、とにかく付けておくおまじないだと思っていてください。

▶▶その他

1行目の「MyC.c」は、このソースコードのファイル名のことです、C言語で書いたソースコードのファイル名は、「これは、C言語のソースコードだよ」ということがわかるように、「**ソースコード名.c**」と書くことにしましょう。

main関数の左にある、「int」や6行目の「return」については、いまはこういうふうに書くのだとおぼえておいてください。

Section 02 文を組み合わせること

ここでは、コンピュータにやらせたいことを、どうやってコンピュータにわからせるか（どうやって文を組み合わせていくか）について、説明していきます。

アルゴリズム

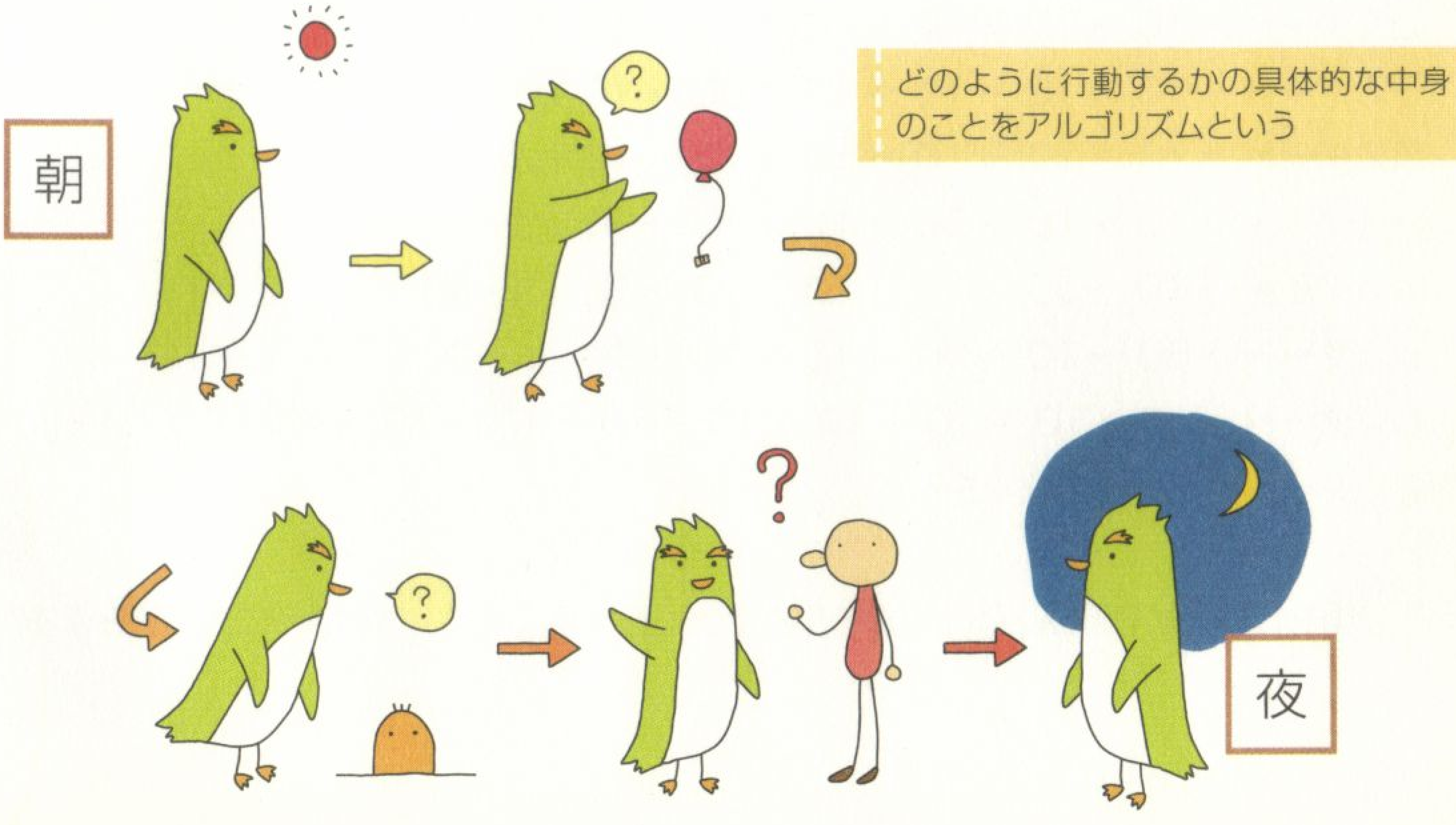
プログラミングすることは、プログラミング言語の文をいくつも組み合わせてコンピュータへの指示を与えることです。逆にいうと、「文の組み合わせで表現できるならば、プログラミング（コンピュータへの指示を与えることが）できる」ということです。

この文の組み合わせを考えていくとき、はじめにこれをやったら次はこれを、その次はこれを、というような手順を考える必要があります。この手順のことを**アルゴリズム**といいます。

▶専門的な説明

人は、ふだんの生活の中で、朝目覚めてから夜寝るまでに様々なことに出会いますが、その都度、それを解決するための処理手順を考えて行動しています。

例えば、遅刻しないですむには、どのように行動するかを考えます。この、どのように行動するかの具体的な中身をアルゴリズムといいます。



どのように行動するかの具体的な中身のことをアルゴリズムという

プログラミング言語をたくさん知っていて、文を作るための規則を理解していても、問題を解決する手順、アルゴリズムができていないと、プログラミングすることができません。

アルゴリズムは、日本語で**算法**と訳されています。

つまり、アルゴリズムとは、ある目的を実現するために必要な作業の手順を、明確に述べたもののことを指します。

これに対して、例えば、一般的にいわれる職人技は、その手順が個人の経験やセンスによるので、アルゴリズムとはいえません。

アルゴリズムのよいところは、その指示に正確に従えば、誰でもまったく同じ結果が得られるということです。

▶▶矢印でつながる図

ここでは、もう少し具体的にアルゴリズムについて説明していきます。

高さの違う3種類の積み木があるとします。1個目は10cm、2個目は50ｃｍ、3個目は100ｃｍです。この、3種類の積み木を組み合わせて、100cmになるようにするには、どうしたらよいのかを、数字と矢印で考えてみることにします。

例えば、次のようになります。

❶スタート→100（ゴール）

このように、次々と、積み上げる積み木の組み合わせを考えていきましょう。

❷スタート→10→10→10→10→10→50（ゴール）
❸スタート→10→10→10→10→50→10（ゴール）
❹スタート→10→10→10→50→10→10（ゴール）
❺スタート→10→10→50→10→10→10（ゴール）
❻スタート→10→50→10→10→10→10（ゴール）
❼スタート→50→10→10→10→10→10（ゴール）
❽スタート→10→10→10→10→10→10→10→10→10→10（ゴール）
❾スタート→50→50（ゴール）

この図を、積み木を積み上げたときの高さがよくわかるように、次のように変えてみます。

100

❶（スタート）0 → 100（ゴール）

10 10 10 10 10 50

❷（スタート）0 → 10 → 20 → 30 → 40 → 50 → 100（ゴール）

⋮

50 50

❾（スタート）0 → 50 → 100（ゴール）

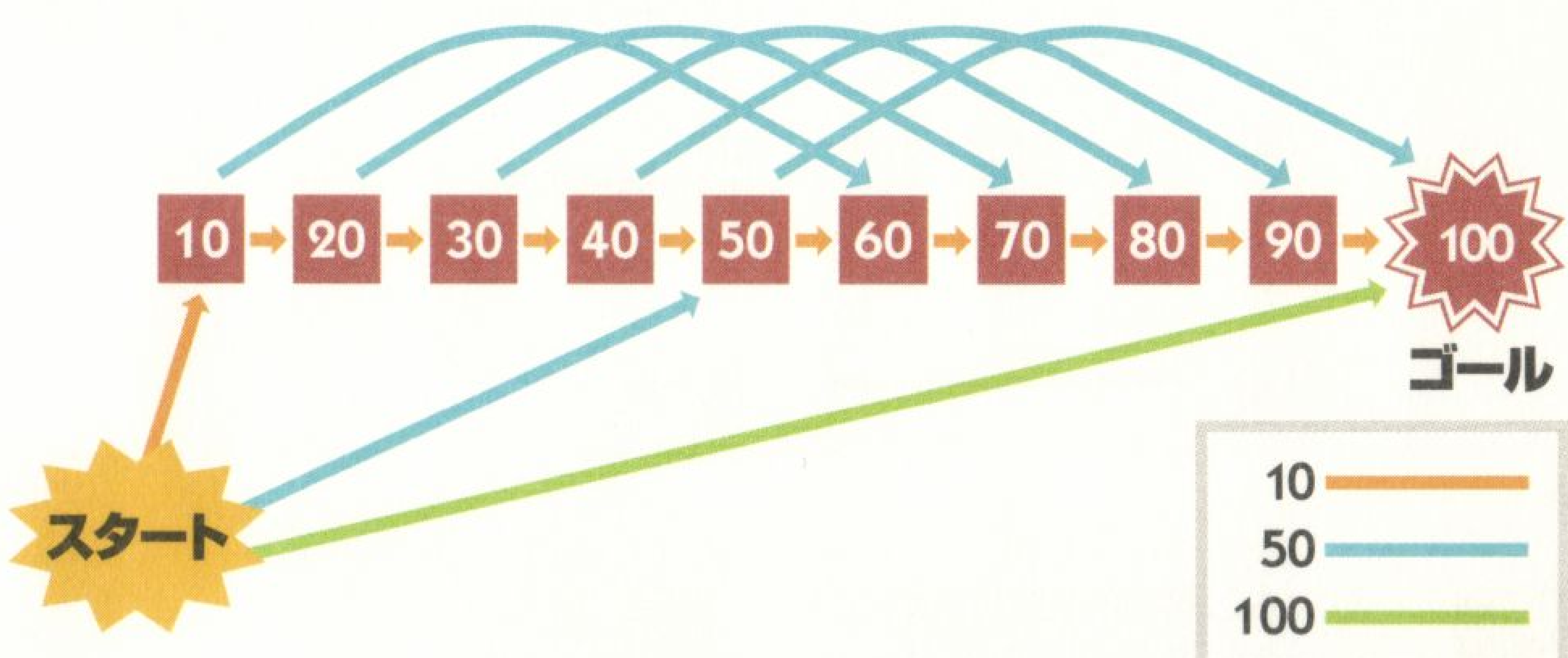
これで、高さがよくわかるようになりましたね。

さて、今度は、この図を1つにまとめてみたいと思います。

▶▶アルゴリズムの図

次の図は、何を表していると思いますか？

実はこの図は、3種類の高さ（10cm、50cm、100cm）を持つ積み木を組み合わせて、高さを100cmにするためのアルゴリズムを示したものだったのです。この図を、プログラミング言語の文法に従って文にしたものが、プログラムの**ソースコード**になるのです。



この図を見れば一目瞭然ですが、コンピュータは、この図を見て理解してくれません。そこで、プログラミングという作業が必要になるのです。

それでは、この図がどのようにして、ソースコードに変わっていくのかを見ていきましょう。

スタート → 0 → 10、50、100
- 10なら10へ
- 50なら50へ
- 100なら100へ

10 → 10、50
- 10なら20へ
- 50なら60へ

20 → 10、50
- 10なら30へ
- 50なら70へ

30 → 10、50
- 10なら40へ
- 50なら80へ

40 → 10、50
- 10なら50へ
- 50なら90へ

50 → 10、50
- 10なら60へ
- 50なら100へ

60 → 10
- 10なら70へ

⋮

100 → ゴール

⬇

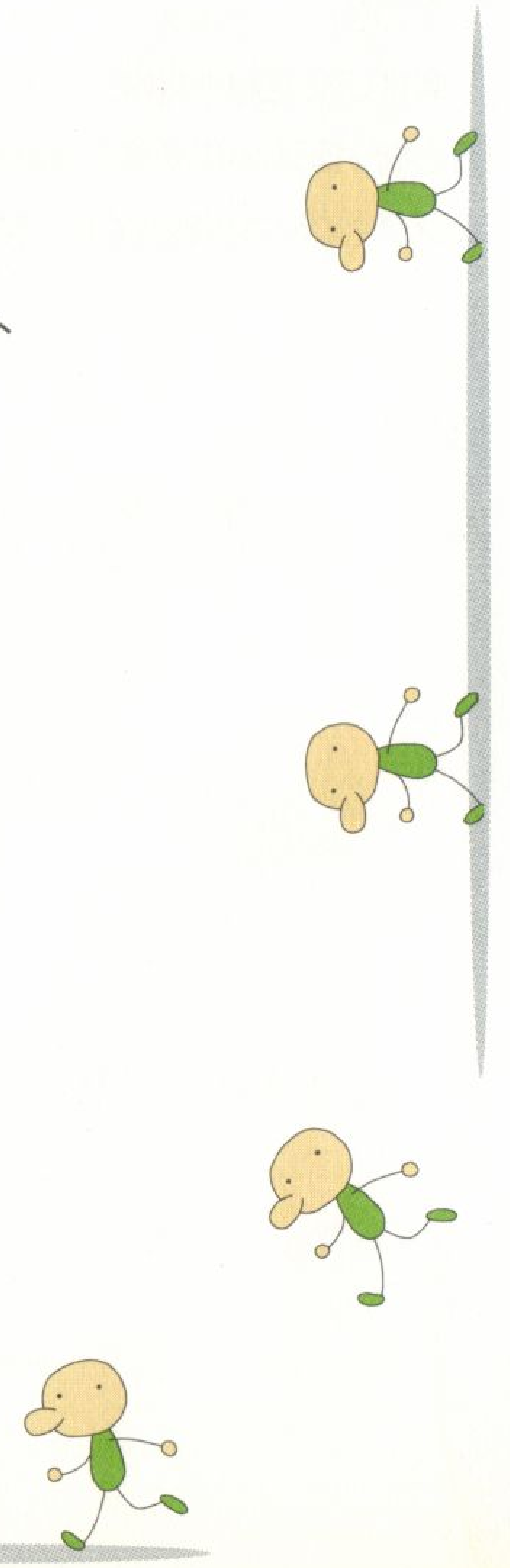
スタート

0:　もし高さ＝10ならば10へ
　　もし高さ＝50ならば50へ
　　もし高さ＝100ならば100へ

10:　もし高さ＝10ならば20へ
　　もし高さ＝50ならば60へ

20:　もし高さ＝10ならば30へ
　　もし高さ＝50ならば70へ

30:　もし高さ＝10ならば40へ
　　もし高さ＝50ならば80へ



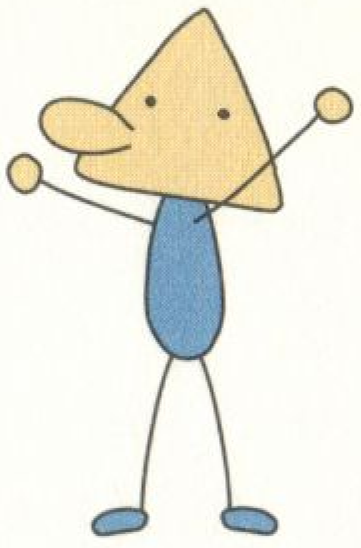
40:　もし高さ＝10ならば50へ
　　 もし高さ＝50ならば90へ

50:　もし高さ＝10ならば60へ
　　 もし高さ＝50ならば100へ

60:　もし高さ＝10ならば70へ

⋮

100:　ゴール



このように、アルゴリズムの図は、プログラムのソースコードに変わっていくことができるのです。

▶▶計算できるものとできないもの

コンピュータは、アルゴリズムの図で表すことができれば、それを計算して結果を出力することができます。

このことから、「世の中の多くのことは、アルゴリズムの図で表すことができる（アルゴリズムが存在する）ので、コンピュータが**解ける問題**（**計算可能／決定可能な問題**）は、たくさんある」ということが、わかると思います。

しかし、ここで気付いてほしいのは、アルゴリズムが存在しない問題をコンピュータは解くことができないということです。こういう問題のことを、**解けない問題**（**計算不可能／決定不可能な問題**）といいます。

また、解ける問題の中でも、おそらく実用上、解くことができない問題（実際に解けない問題）もあるのです。ちょっと難しいのですが、解けたころには、宇宙がなくなっているとか、そんな時間をかけてしか解けない問題もあるのです。例えば、「素因数分解※」や「ナップサック問題※」という、有名な**実際には解けない問題**があります。

簡単にいってしまうと、いまのコンピュータには、やらせたくても無理なことがあるということです。

※**素因数分解**　素数のかけ算のかたちにすること。例 6=2×3
※**ナップサック問題**　複数の品物（それぞれ、サイズ値段が違う）が与えられたとき、ナップサックにおさまり、合計の値段が最大になる品物の組み合わせを考える問題。

例題 アルゴリズム❶

次のうち、手順を詳しく書けばアルゴリズムとして説明できそうなものはどれですか？

❶ ギャンブルでの勝ち方

❷ TVゲームのクリアの仕方

❸ おいしいパンの作り方



●解答

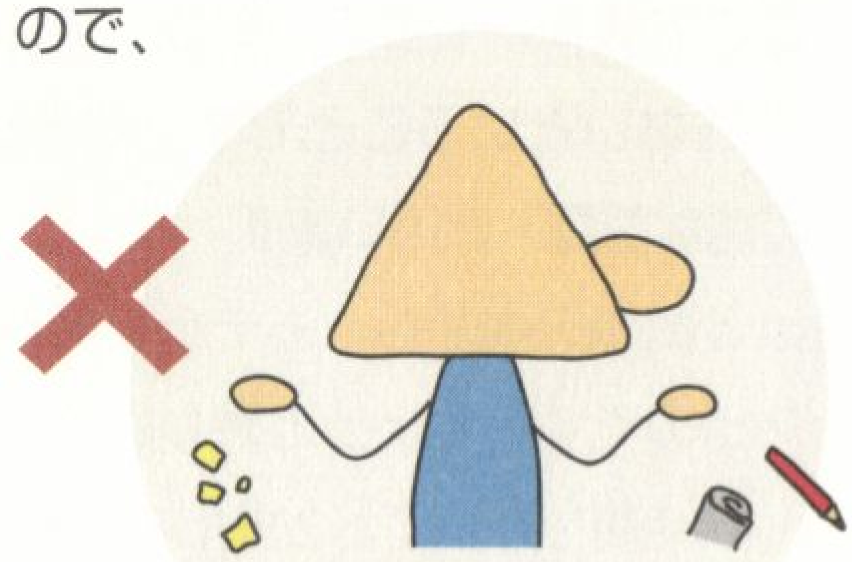
❷

●解説

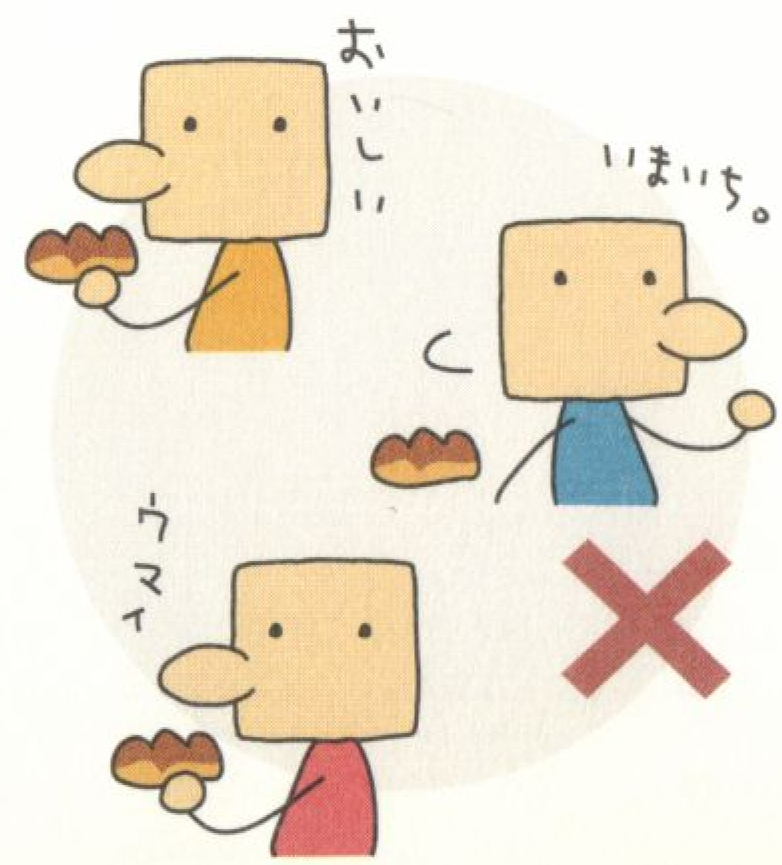
❷の「TVゲームのクリアの仕方」は、攻略の手順を明らかにできますし、その手順どおりにプレーすれば、必ず攻略できます。

❶の「ギャンブルの勝ち方」は、確率統計手法で多少の予測はつきますが、実際には勝つか負けるかは時の運です。誰がどんなときでも、必ず勝てる方法はありません。

❸の「おいしいパンの作り方」は、パンの作り方には手順がありますが、「おいしい」の感じ方が人それぞれ違う（数値化できない）ので、おいしくするための手順がわかりません。







例題 アルゴリズム❷

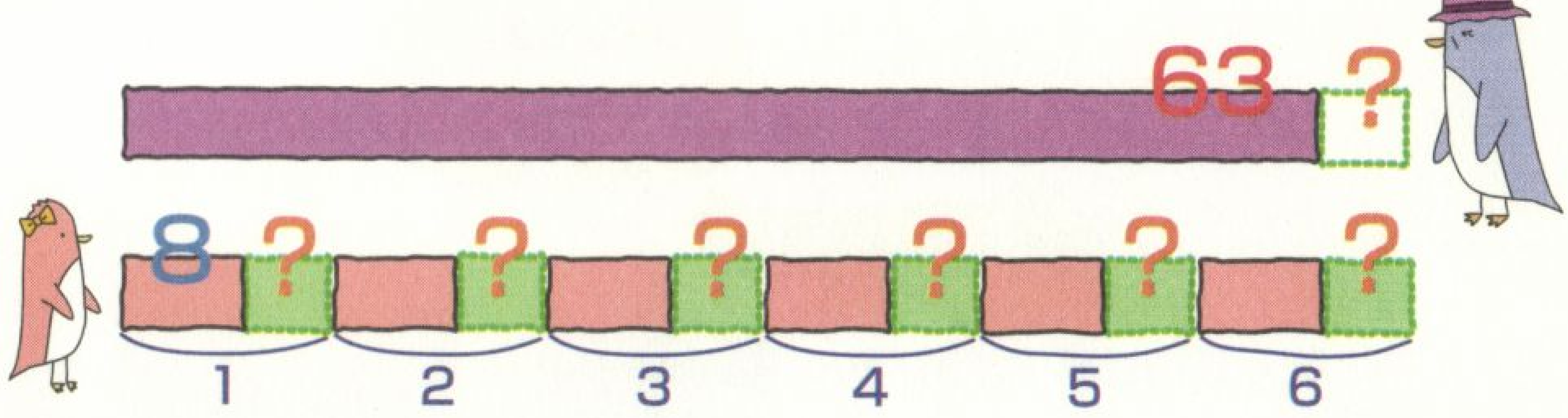
次の問題を解くアルゴリズムを考えて、その手順を文で説明してください。

A子さんは、8歳で、そのおばあちゃんは、63歳です。おばあちゃんの年齢が、A子さんの年齢の6倍になるのは、何年後になりますか？



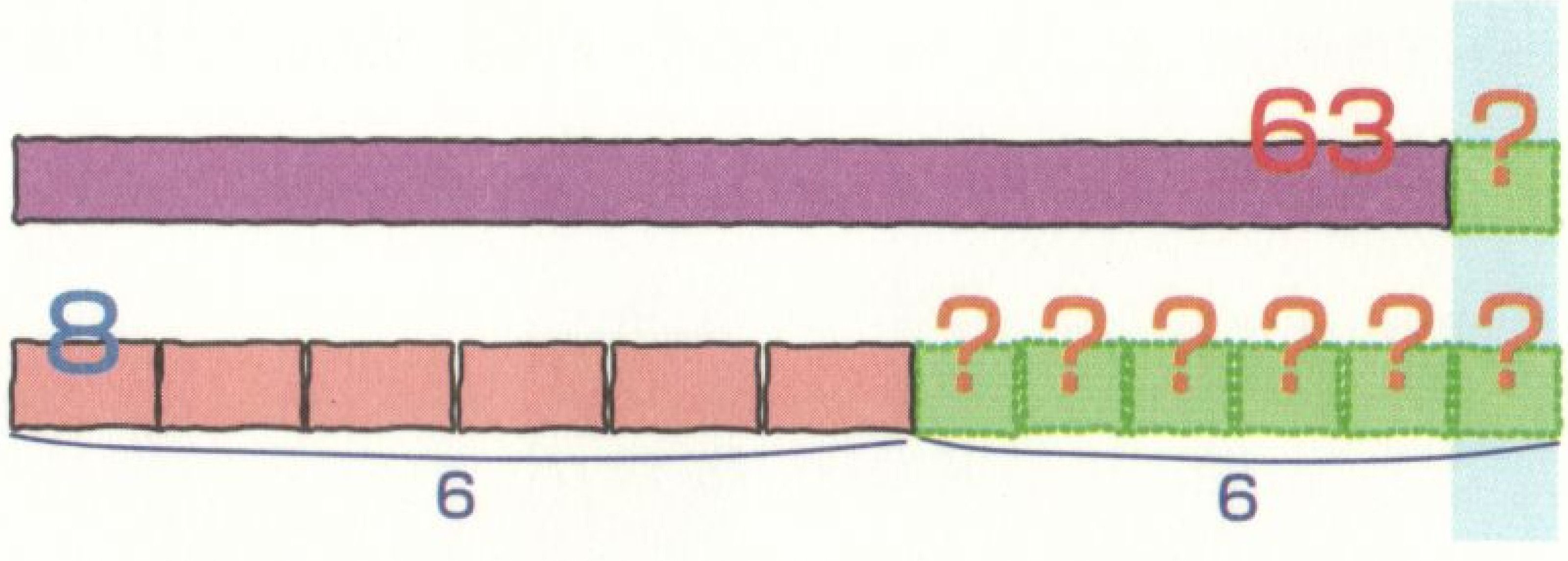
●解答と解説

図を描いて考えてみましょう。

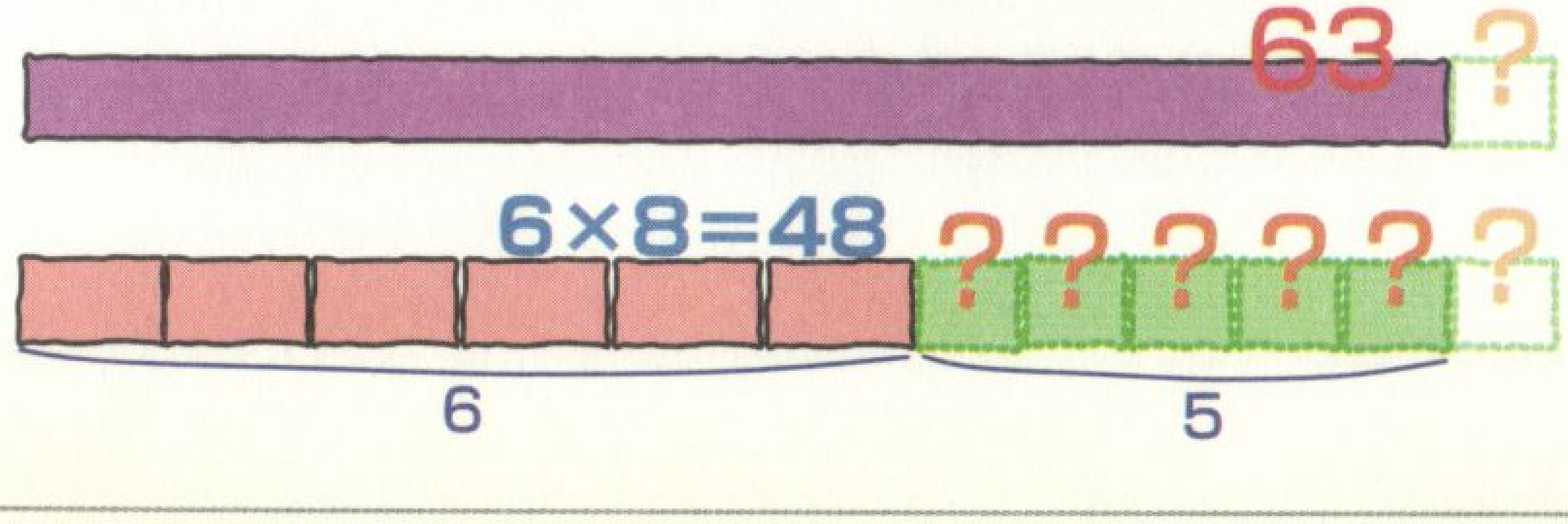


図を見ると、現在のA子さんの年齢（8）に「?」をたしたものの6倍と、おばあさんの年齢（63）に「?」をたしたものが等しくなっています。

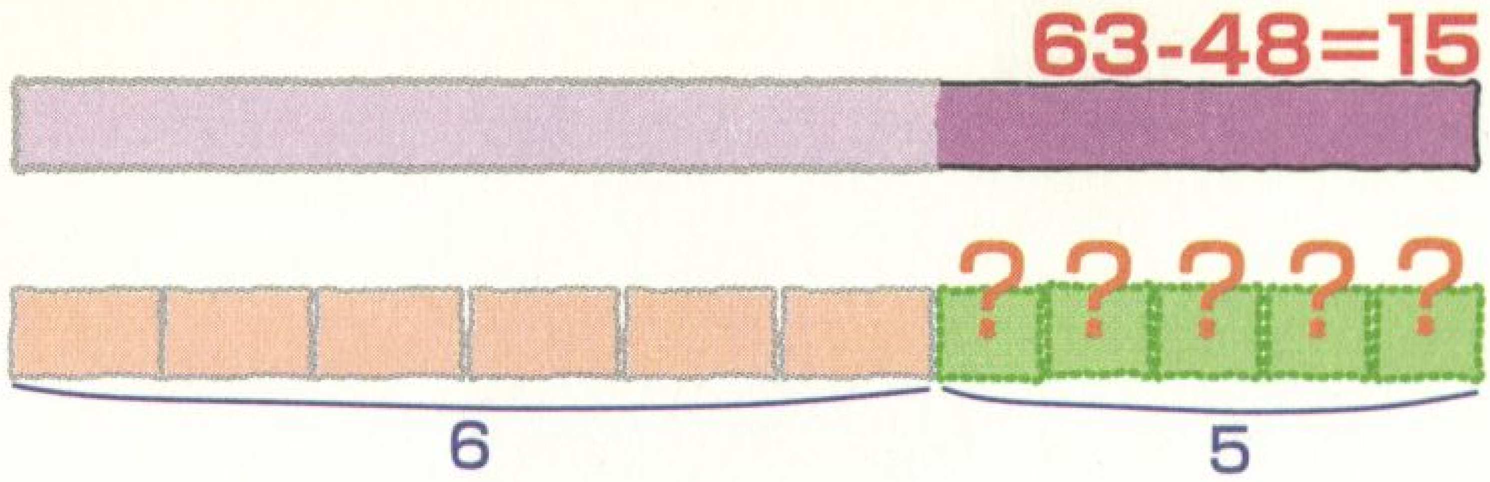
ここで図を次のようにしてみます。



右端の「?」がA子さんとおばあちゃんでダブっていますね。そこで両方から消したのが次の図です。



もっとすっきりさせると、こうなります。



このことを、文にすればよいわけです。

手順❶：A子さんの現在の年齢を、6倍します。6×8＝48になります。
手順❷：現在のおばあちゃんの年齢から48を引きます。63−48＝15になります。
手順❸：？の5つ分が、15ですので、15を5で割ってやります。
　　15÷5＝3（年後）



COLUMN 数当てゲーム その1

相手の思っている数字を当てるゲームをしてみましょう。
3枚のカードを用意してください。それぞれのカードには、次のように数が書かれています。

1枚目：1、3、5、7
2枚目：2、3、6、7
3枚目：4、5、6、7

ゲームのやり方は、1枚目から3枚目のカードを相手に順に見せていって、相手の思っている数字が、各カードに「あるか、ないか」を質問していきます。
相手の思っている数字を当てるには、「ある」と答えたカードの先頭の数字をたしていけばよいのです。試してみてください。
どうして、これで数を当てられるかわかりますか？
（本文83ページ「数当てゲームその2」へ続く）

COLUMN C言語なQ&A

Q C言語のプログラムがスラスラ書けるようになるにはどのぐらいの時間がかかりますか？

A 趣味として、スラスラ書けるようになるには、いままでにプログラミングの経験がないとすると、160時間ぐらいかかると考えてください。1日に1時間、プログラミングを勉強するとして、半年ぐらいはかかると思います。
　ふつうに使われている、どのプログラミング言語でも、このスラスラ書けるようになるために必要な時間は変わりません。

Q プログラミング教室に通おうと思っていますが、役に立ちますか？

A プログラミング教室というものが、あるかないかは別として、やる気次第というべきでしょう。1冊の本でも、プログラミングする気さえあれば、問題ありません。
　逆にいうと、プログラミングを嫌々するのならば、教室に通っても、役に立たないと思います。習うより、慣れろです。

Q プログラマーになりたいのですが、何か資格とか必要ですか？

A プログラマーになるのに、資格や免許はいりません。それは、職業プログラマーであったとしても同じです。ビル・ゲイツ氏が、資格を持っているとは、聞いたことないですし…。
　ただ、そういった認定資格試験を受けることによって、自分の技術力を確認できると思います。また、資格があると、給料が多少ですが高くなるとか、就職に有利になるかもしれません。

Q 英語の文法はよくわからないのですが、C言語の文法はわかりますか？

A 英語や日本語などと違って、プログラミング言語の文法は限られています。そのためおぼえなければならないことや、わからなければならないことが多くありませんので、文法もすぐわかると思います。

Q C言語の用語はすべて暗記しないとダメなのですか？

A プログラミングは、数学の証明問題を解くのと同じです。暗記というものは必要ありません。忘れたら、本を参照にすればよいのです。

Q 何冊ぐらい本を読めば、プログラムはわかるようになりますか？

A プログラミング言語の本は、そのことがプログラミング覚えることを難しくしているといってもよいぐらいに、本当にたくさんありますね。
　でも、一つのプログラミング言語を覚えるのには、読むのは1冊で十分です。そして、本棚に揃えておくには、3冊ぐらいあればよいでしょう。
　1冊目は、プログラミングとは何であるかを知るための本（入門編）、2冊目は、覚えようとするプログラミング言語で、できること（機能）や考え方（概念）、文法を解説した本（実践編）、3冊目は、実際に、プログラム作成するときに必要になる知識を教えてくれる本（応用編）、と考えればよいでしょう。

practice 練習問題

解答は213ページ

1 ソースコード　難易度★★☆☆☆

この章で説明したソースコード「MyC.c」のクラス名を、「MyStudy」に変更して、「MyStudy」というファイル名で保存してコンパイルを実行してください。

▶▶ヒント：本文43ページ参照

2 アルゴリズム　難易度★★★★☆

自宅から、目的地（旅行先や、学校や会社など）までの道のりを、交通機関で区別して、手順として書いてみてください。

例）　自宅から歩いて五条駅まで行き、そこから地下鉄に乗って京都駅まで行きます。京都駅から東京駅まで新幹線に乗り、東京駅から新宿まで中央線に乗ります。新宿から△大学に歩いて行く場合。

手順❶：自宅から五条駅 徒歩
手順❷：五条駅から京都駅 地下鉄
手順❸：京都駅から東京駅 新幹線
手順❹：東京駅から新宿駅 中央線
手順❺：新宿駅から△大学 徒歩

▶▶ヒント：本文59ページ参照

3 main関数　難易度★★★★☆

C言語のソースコードの「main関数」は特別な関数ですが、どのように特別かを説明してください。

▶▶ヒント：本文58ページ参照

memo

Chapter3のまとめ

★関数

関数とは、コンピュータにしてもらいたいことを文やブロックで書いたものをいいます。

★コメント

コメントとは、ソースコードの中に書くメモのことです。

★main関数

「main関数」とは、特別な関数です。

C言語のソースコードには、必ず「main関数」が必要です。

C言語のプログラムは、この「main関数」から処理が始まるようになっています。

★予約語

予約語とは、プログラムの実行順序の制御や変数の型など、C言語があらかじめ用意している単語のことです。

★プリプロセッサ制御文

プリプロセッサ制御文とは、「#」がついた行のことをいいます。

★アルゴリズム

アルゴリズムとは、与えられた問題を解く手順のことです。

★アルゴリズムであるための条件

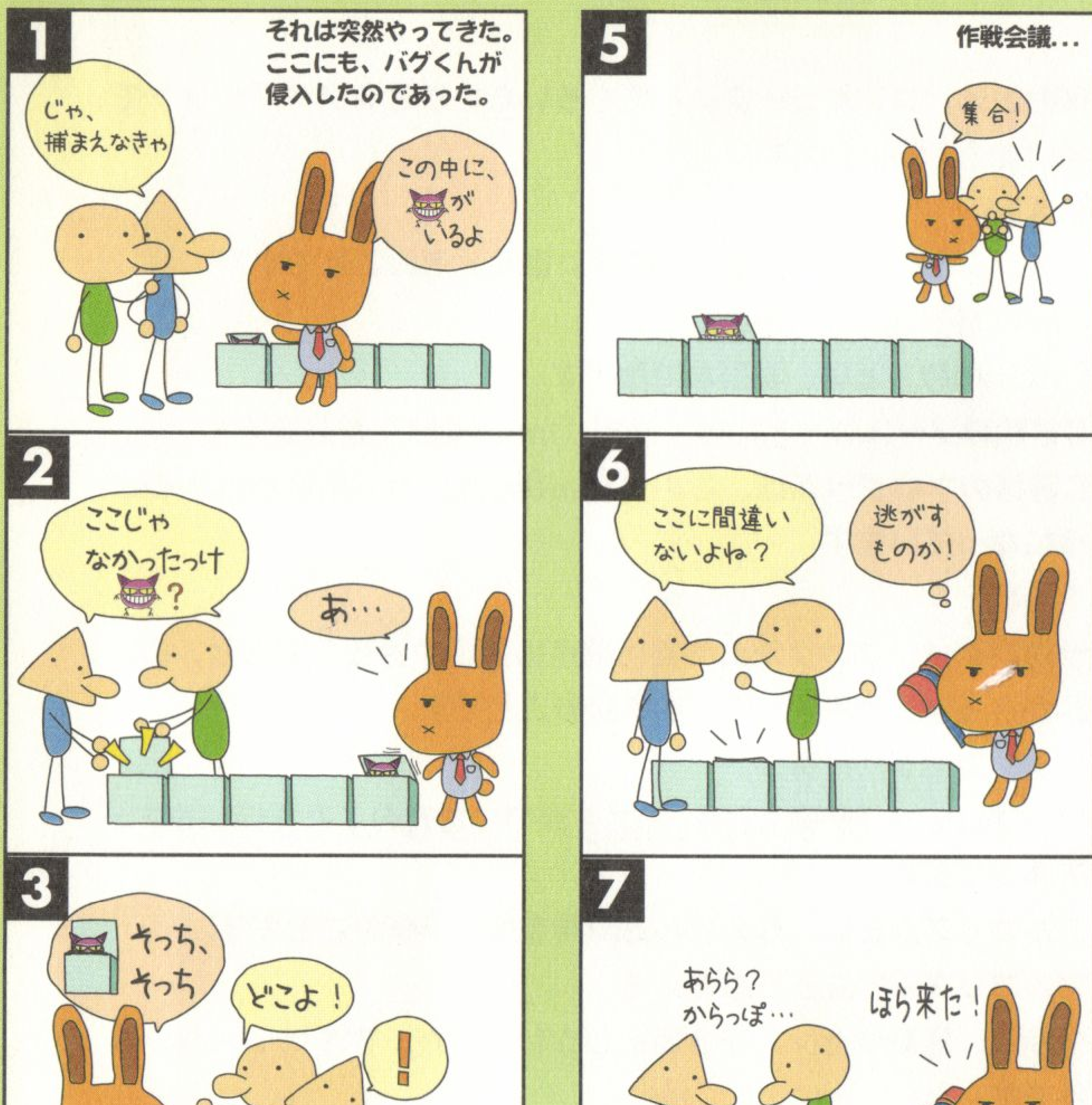
手順に、経験やセンスを必要としないことが、アルゴリズムの条件です。

★コンピュータと問題の関係

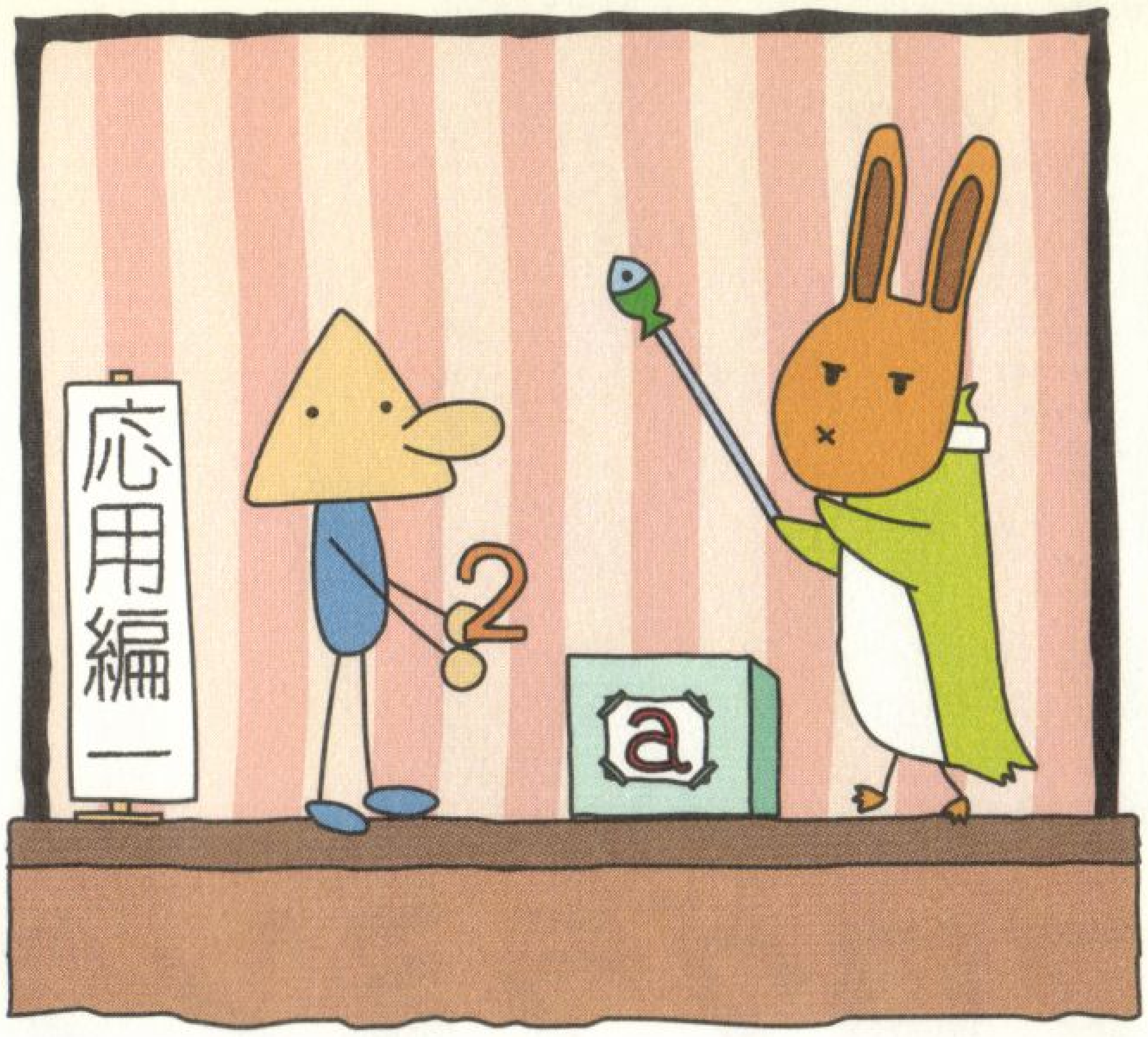
コンピューターは、解ける問題、解けない問題、実際に解けない問題があります。

トキメキ4コマ

バグくんをつかまえろ①



92ページに続く



★いよいよここからC言語のプログラミングです。

まずは変数や文字を使って、

簡単な計算にチャレンジしてみましょう。

★C言語の制御文を理解したい方は

➡第5章本文94ページへ

★C言語のより高度なプログラミングを学びたい方は

➡第6章本文138ページへ

Chapter 4

プログラミングの心得❶

01 データを入れておく箱 ― 変数

- 変数
  - 変数の名前
  - 変数の種類
  - 変数の宣言
  - 変数への代入
  - 変数を使う

02 計算するための記号 ― 演算子

- 演算子



この章でマスターすること

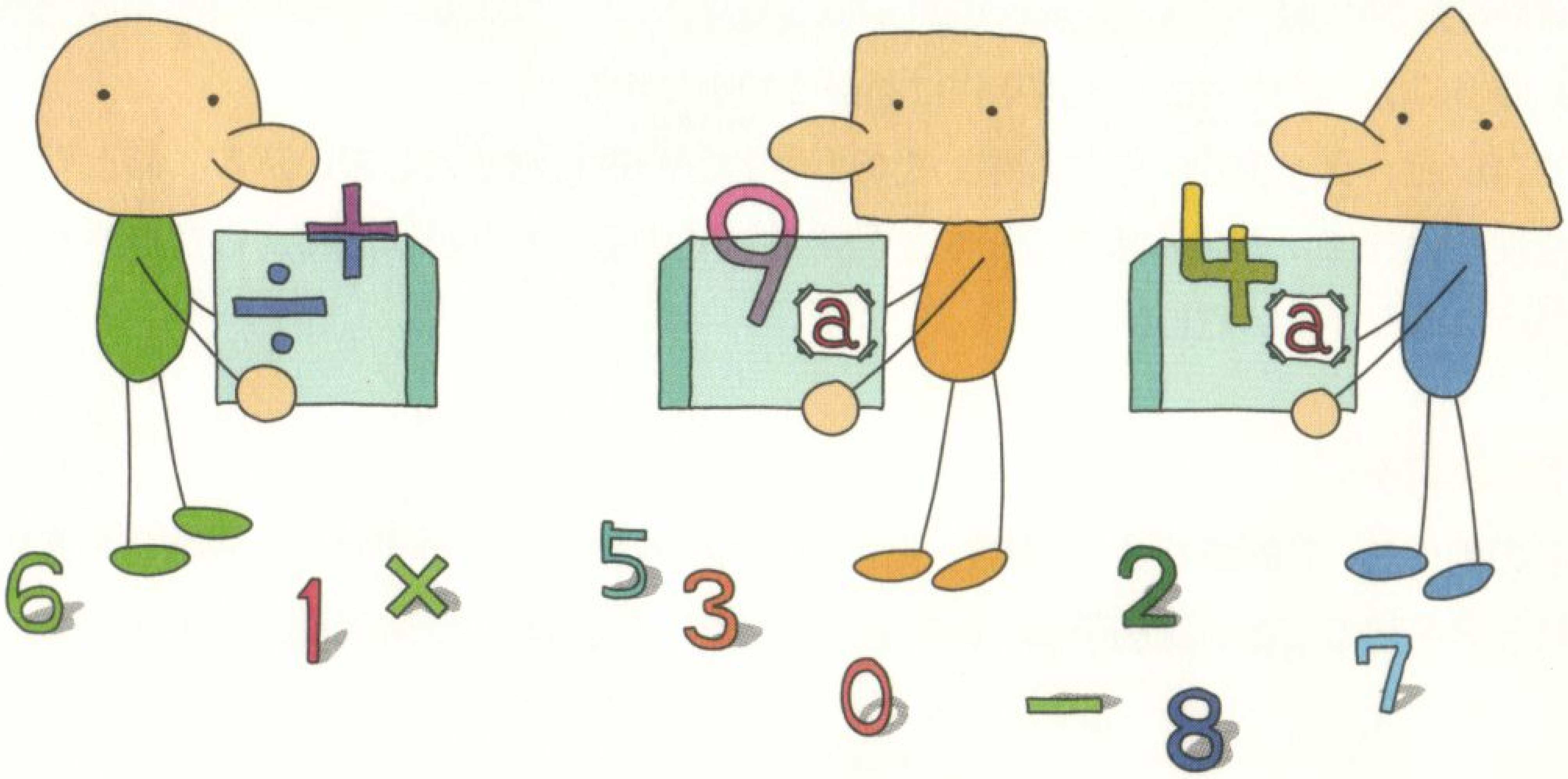
ここでは、プログラミングでは必ず使うことになる**変数**と、それを計算するための記号である**演算子**について説明します。変数は数学で習いましたね。また、演算子は、足し算や引き算で使う「+」や「−」などでおなじみの記号です。

ただし、人とコンピュータがコミュニケーションするために作られたプログラミング言語は、算数や数学など、人と人とがコミュニケーションをとるための自然言語とは違った考え方（概念）を必要とします。

例えば、変数の代入です。「代入」とは、C言語ではいったいどんなものでしょうか？

この章では、変数と演算子の使い方を学んでいきましょう。

算数や数学で習った演算子とはどこが違う？



C言語必須用語

- 変数
- データ型
- 変数の宣言と初期化
- 演算子

ステップアップ用語

- 識別子
- 代入
- 式

Section 01

データを入れておく箱 ―― 変数

変数とは何でしょうか？　簡単にいえば、文字や数字を入れておく箱のようなものです。ここでは、データを入れておく箱「変数」についてマスターしましょう。

変数

プログラムは、データを保存しておく場所を、確保しなければなりません。このデータを保存しておく場所のことを**変数**といいます。これは、人が領収書（データ）を机の中（変数）に集めておいて、あとで計算するのと同じことです。



変数を使うと、コンピュータの記憶領域（メモリ）に、データをおぼえておくための場所を確保できます。

▶▶変数の名前

変数を使うには、その変数に名前を付ける必要があります。

そうでないと、データをどこに記憶したのかがわからなくなってしまいます。あとでその変数を使うときに、どのようなデータが記憶されているかわかるように、わかりやすい名前を付けたほうがよいでしょう。

▶専門的な説明

変数名の他にも、C言語では、「関数」に名前を付けなければなりません。変数名も含めて、これらの名前のことを**識別子**といいます。識別子の名前の付け方は変数と同じです。



- 名前には半角の英数字を使う。　→ a,b,a1,b1 ○
- 数字で始まる言葉は使えません。　→ 例：1a ×
- 全角文字は使えません。　→ 変数A ×
- mainのようにあらかじめC言語にある言葉は使えません。　→ main ×

▶▶変数の種類

プログラムは、様々な種類のデータを扱います。例えば、給料を計算するソフトならば、社員の名前には「文字」を、金額の計算には「整数」や「小数」を扱います。このようなデータの種類のことを、**データ型**といいます。

データを記憶するための場所である変数も、このデータ型に合わせて用意されています。データ型の違いは、そのままメモリ上に確保される場所の大きさの違いです。

▼C言語でよく使うデータ型

タイプ	型名	サイズ	意味	範囲
整数型	short	16	単長整数	-32,768〜32,767
	int	16	整数	-32,768〜32,767
	long	32	倍長整数	約±21億4700万
浮動小数点型	float	32	単精度浮動小数点数	有効桁7
	double	64	倍精度浮動小数点数	有効桁16
文字型	char	8	文字	8ビットの文字コード

整数型：　　整数のデータ。
浮動小数点型：小数点が付いたデータ。
文字型：　　文字のデータ。

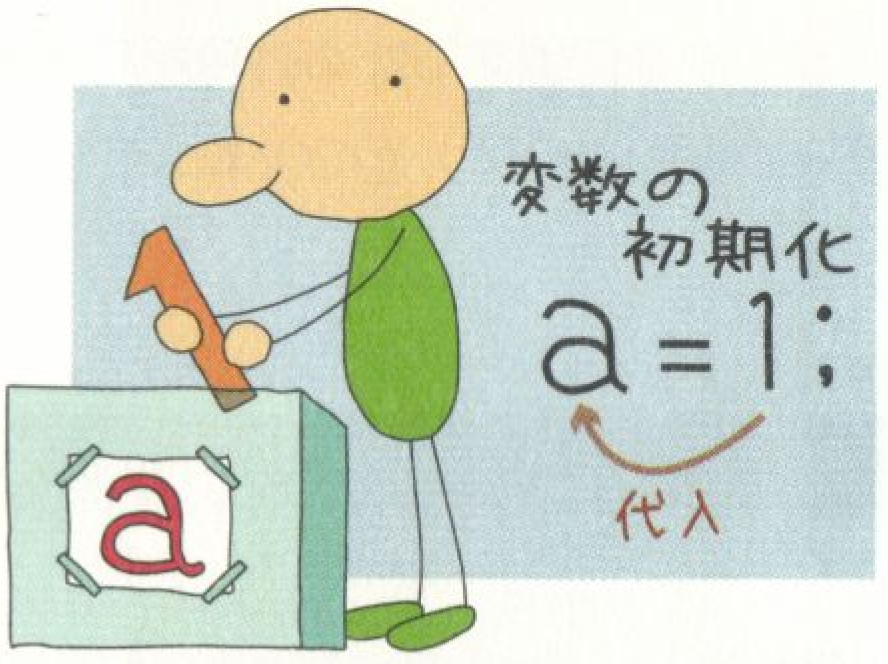


▶▶変数の宣言

変数を使う前には、ソースコードの中で名前とデータ型を指定します。これを**変数の宣言**といいます。プログラムはコンピュータに、変数の宣言によって「これから、変数を使うので準備してください」と指示しているのです。

▶▶変数への代入

変数が準備できたら、変数にデータを記憶することができます 。これには、「=」を使います。算数や数学では、「=」は両辺の値が等しいことを表す記号（等号）ですが、C言語では、右のデータを左の変数に代入することに使われます。



このような代入を表す式を**代入式**といいます。

変数に記憶されたデータ（変数の中身）のことを**変数の値**といいます。

変数宣言と同時に、変数に値を入れることを**変数の初期化**といいます。また、変数は値を代入しなければ使えません。一度データを代入した変数に、再度データを代入することもできますし、変数に変数の値を代入することもできます。

▶▶変数を使う

それでは、実際に変数を使ったソースコードを見てみましょう。本文56ページのソースコードに、変数を使ってみました。

```
/* MyC2.c */
#include <stdio.h>

int main(void)
{
  int number;

  number = 2;
  printf("My C Programming%d¥n",number);
  return 0;
}
```

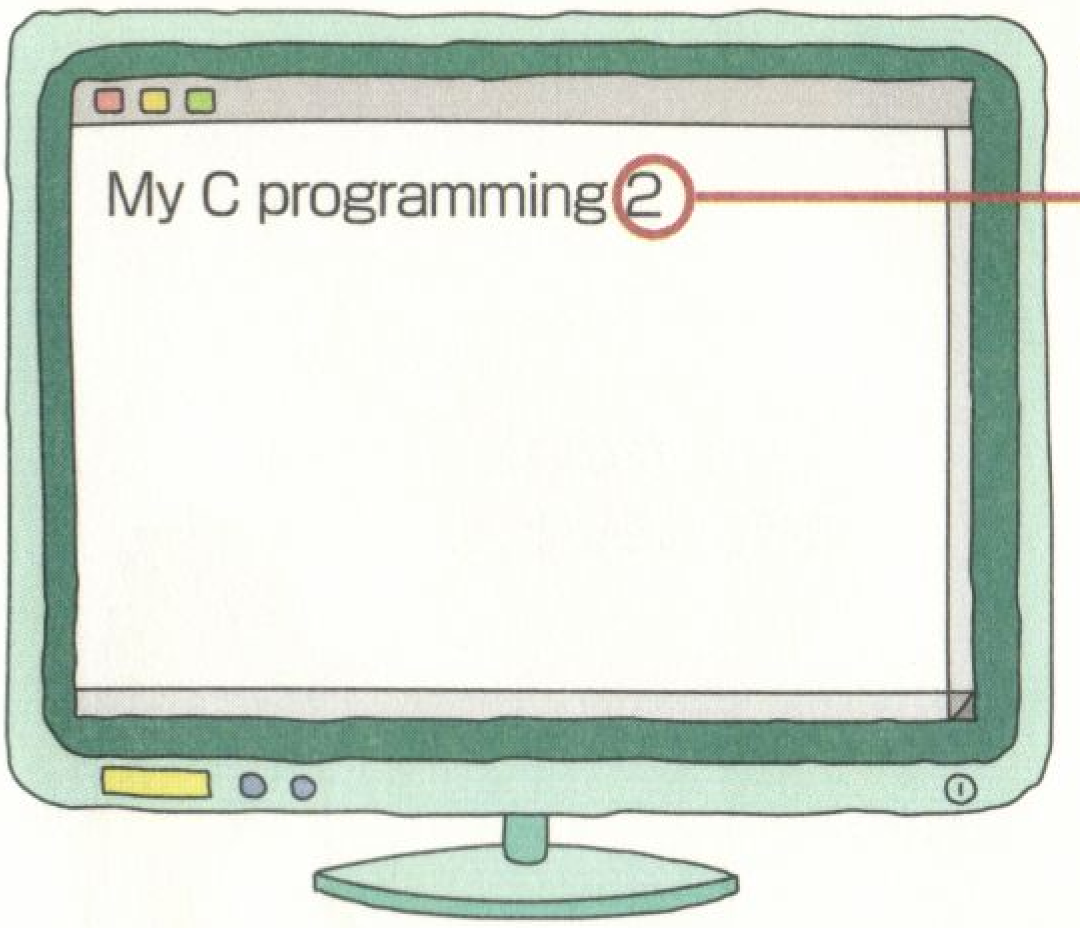
int number; → int型 変数の宣言（宣言文）

number = 2; → 変数の初期化と代入（代入式）

printf(...); → numberの値が、「%d」のある位置に表示されます

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷ 「MyC」フォルダにファイル名「MyC2.c」で保存します。
❸ MS-DOSプロンプトを起動します。
❹ 「**cd c:¥MyC** Enter」と入力してください。
❺ 「**gcc MyC2.c -o MyC2** Enter」と入力します。
❻ 「**MyC2** Enter」と入力します。



◀実行結果

変数の初期化と代入が行われて値が出力される

6行目で変数の宣言（宣言文）を、8行目で変数の初期化と代入を行っています。つまり、8行目の「number=2」は代入式です。

9行目を見てください。サンプルソースコード「MyC.c」の5行目とは違っていますね。この「%d」は、「,」のあとに続くint型の変数「number」の値（中身）を画面に表示するときに、表示したい場所に埋め込みます。「¥n」は、**復帰改行**をするための約束ごとです。

「復帰改行」といっても何のことかよくわからないと思いますので、実際に「¥n」があるときと、ないときとでは、どのような違いがあるのかを実際に見てみましょう。

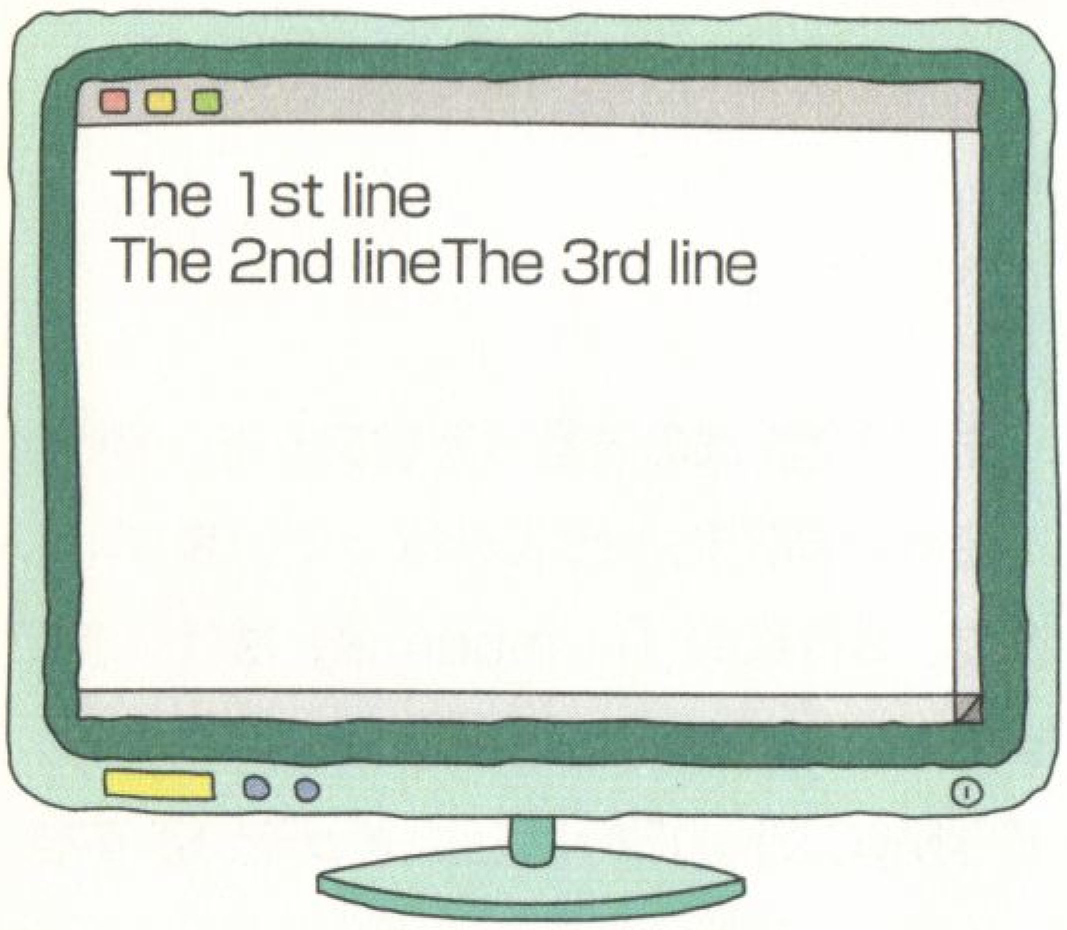
```
/* Footnote8.C */
#include <stdio.h>

int main(void)
{
  printf("The 1st line ¥n");  →「¥n」がある
  printf("The 2nd line");     →「¥n」がない
  printf("The 3rd line ¥n");
```

```
 9:
10:   return 0;
11:
12:};
```

6行目と7行目を見てください。6行目には「¥n」がありますが、7行目にはありません。このプログラムを実行すると、次のようになります。

どうですか、「¥n」の有無の違いがわかりましたか。





●**変数の使い方のまとめ**●

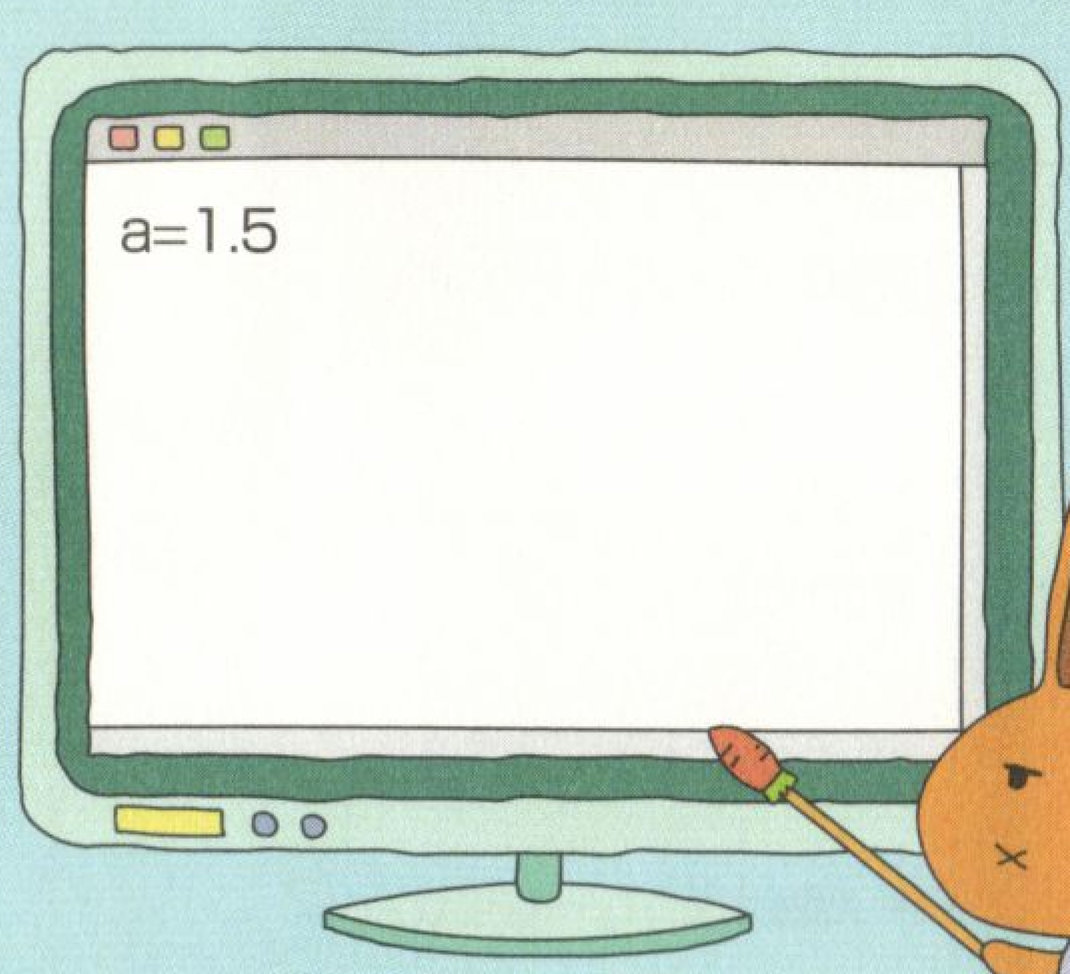
❶ **変数の宣言 ➡ int a;**

❷ **変数の代入 ➡ a = XX;**

❸ **変数の利用 ➡ printf("%d¥n",a);**

（画面にint型の変数「a」の値を出力します）

例題 変数のデータを画面に表示する



float型の変数「a」に、値「1.5」を代入し、その結果を次のように画面に出力するソースコードを作成してください。



●解答

```
/* Exercise1.c */
#include <stdio.h>

int main(void)
{
  float a;

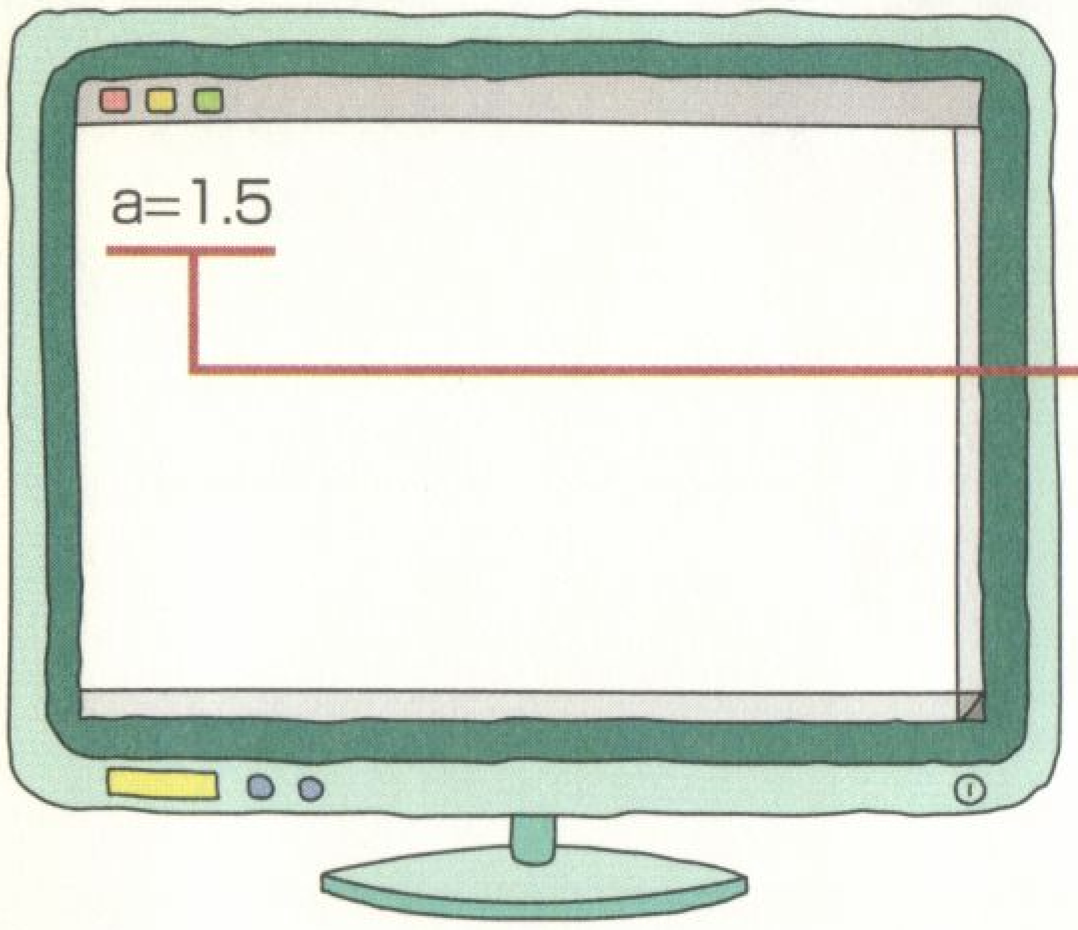
  a = 1.5;
  printf("a = %f¥n",a);    ――→「%f」に注目
  return 0;
}
```

●解説

ここで、もう一度、作成手順と実行結果についておさらいしておきましょう。

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「Exercise1.c」で保存します。
❸ MS-DOSプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc Exercise1.c -o Exercise1** Enter」と入力します。
❻「**Exercise1** Enter」と入力します。

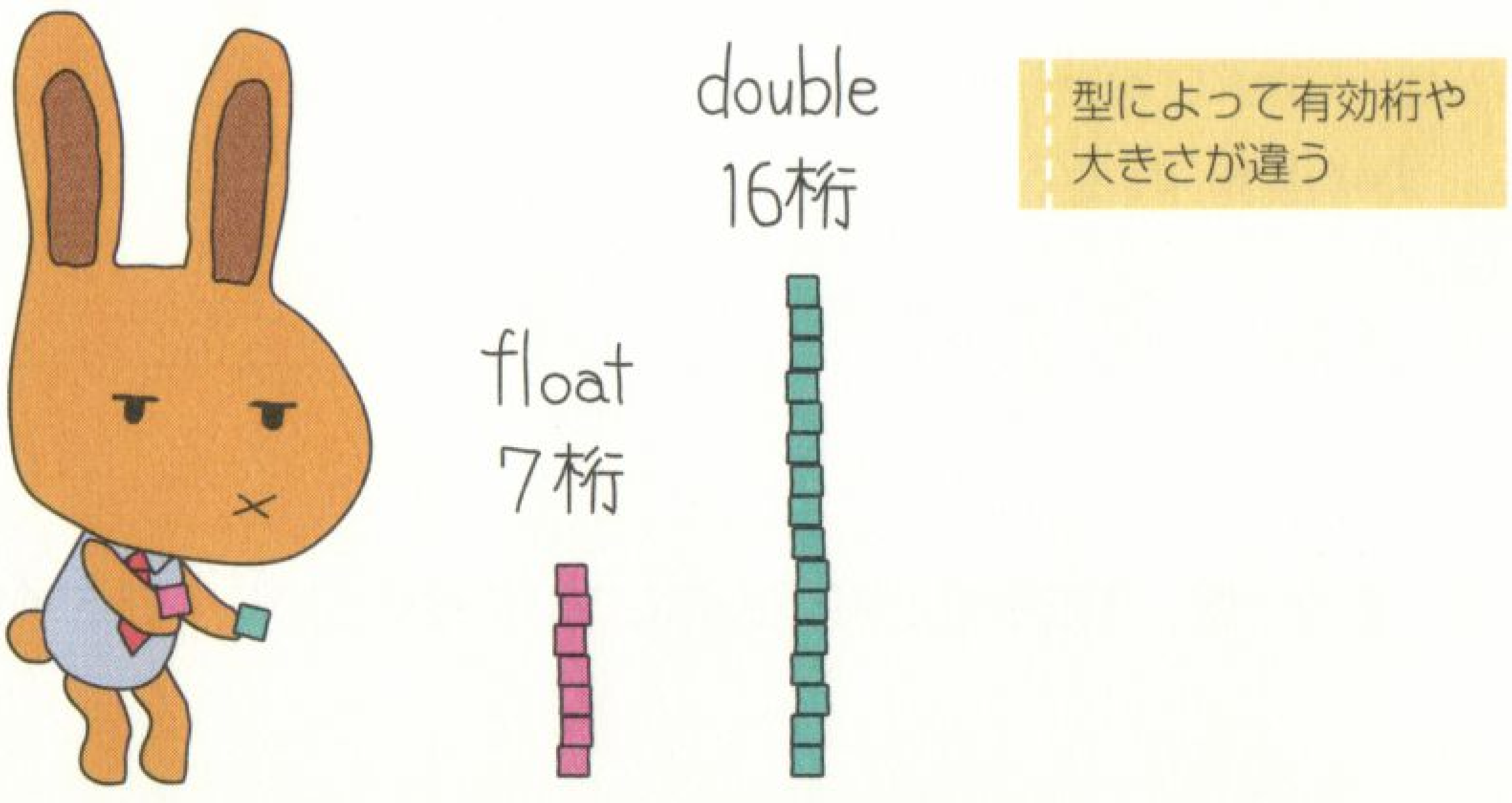


◀実行結果

6行目を見てください。変数aを、「float」型で宣言することで、小数点以下の数値を表せるようにしました。

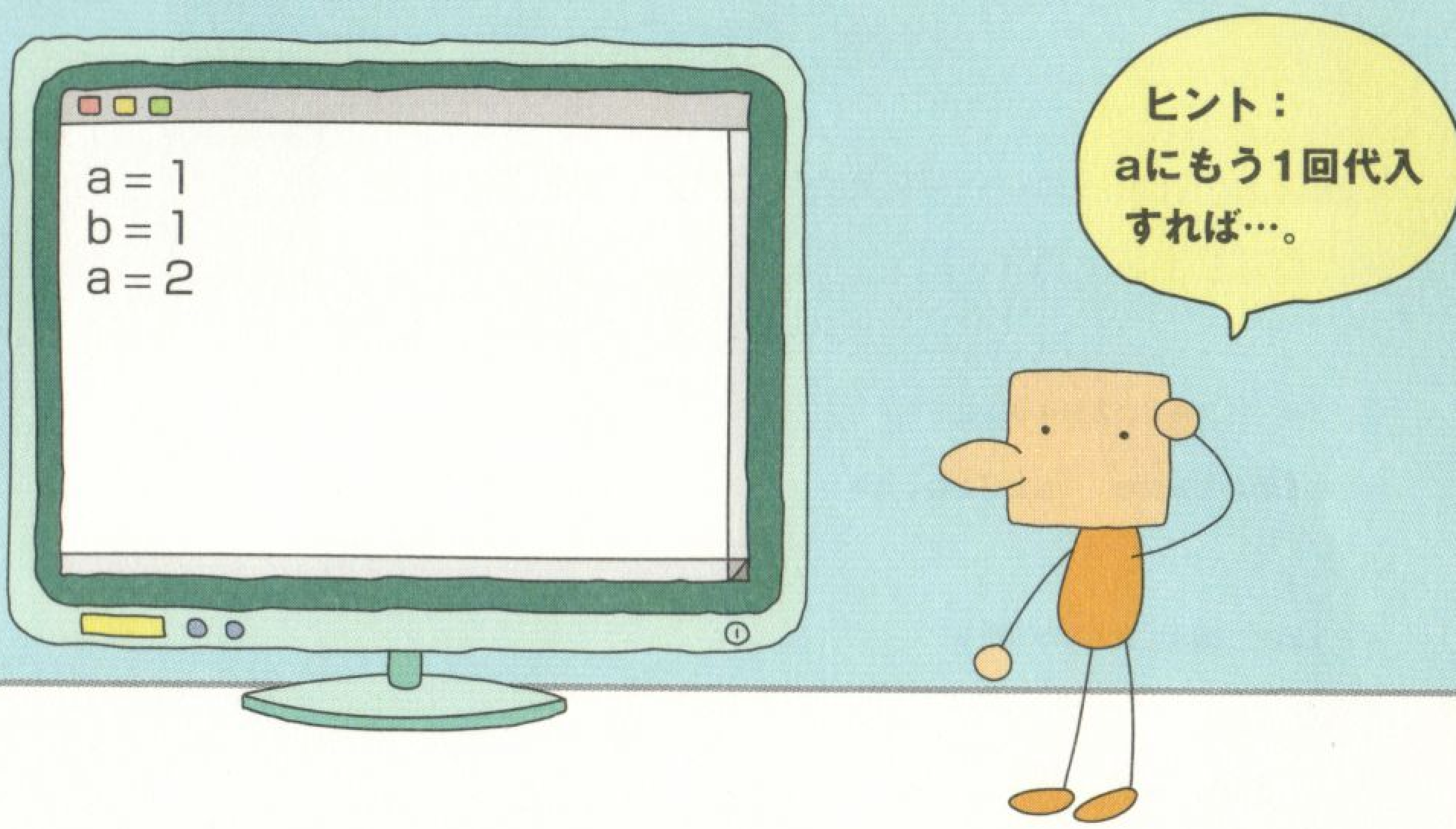
9行目の「%f」は、「,」のあとに続くfloat型の変数「a」の値や、double型の変数の値を画面に表示するときに、表示したい場所に埋め込みます。

「float」型と「double」型の違いは、「本文75ページでの表ーC言語でよく使うデータ型」を見ればわかるように、扱える桁数にあります。



例題 変数の代入いろいろ

次のように画面に出力するソースコードを、変数aの値を変数bに代入し、変数aに新しい値を代入することで作成してください。



●解説

```
/* Exercise2.c */
#include <stdio.h>

int main(void)
{
    int a;
    int b;

    a = 1;
    printf("a = %d¥n",a);
    b = a;    → 変数aの値を変数bに代入
    printf("b = %d¥n",b);
    a = 2;    → 変数aに、新たな値「2」を代入
    printf("a = %d¥n",a);

    return 0;
}
```

●解説

11行目のようにすると、変数に、変数の値を代入することができます。また、13行目では、変数に値を再代入しています。

例題 変数の初期化

次のようなソースコードを入力して実行してみてください。
どうなりましたか？

```
/* Exercise3.c */
#include <stdio.h>

int main(void)
{
  int a;

  printf("a = %d",a);
  return 0;
}
```

●解答

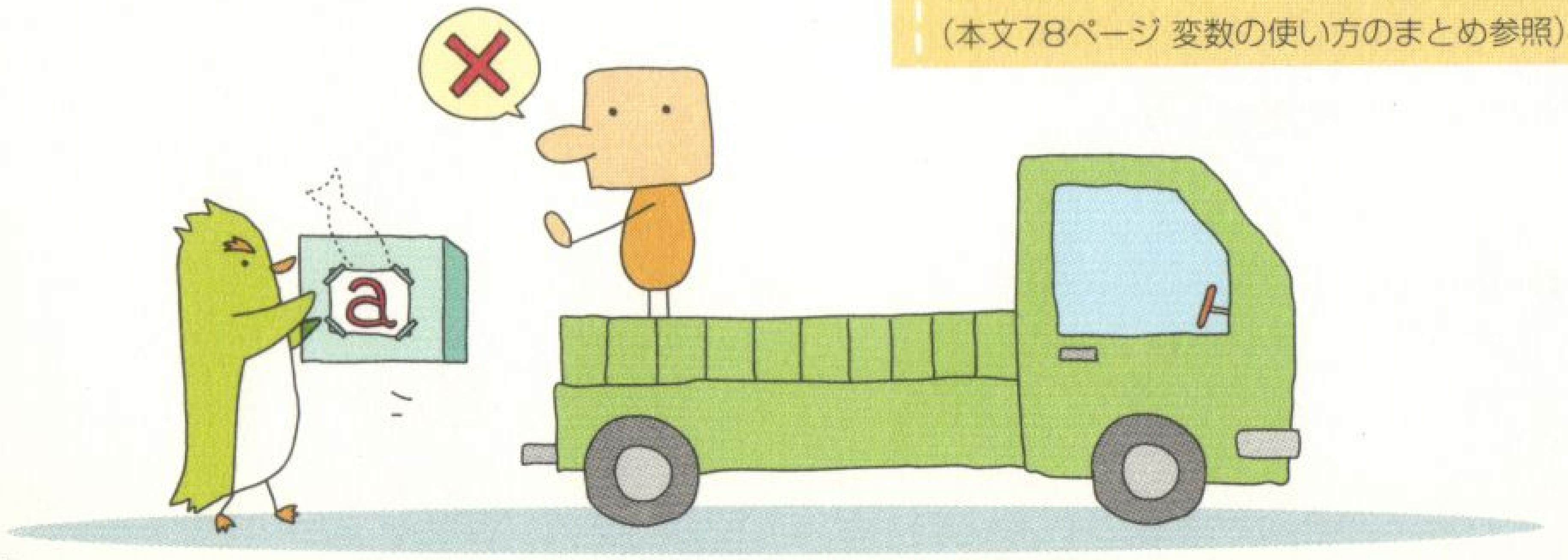
変数に値を代入していないので、おかしな値を表示します。



●解説

変数は値を代入して使うものですから、このように何も代入しないで使うことはできません。

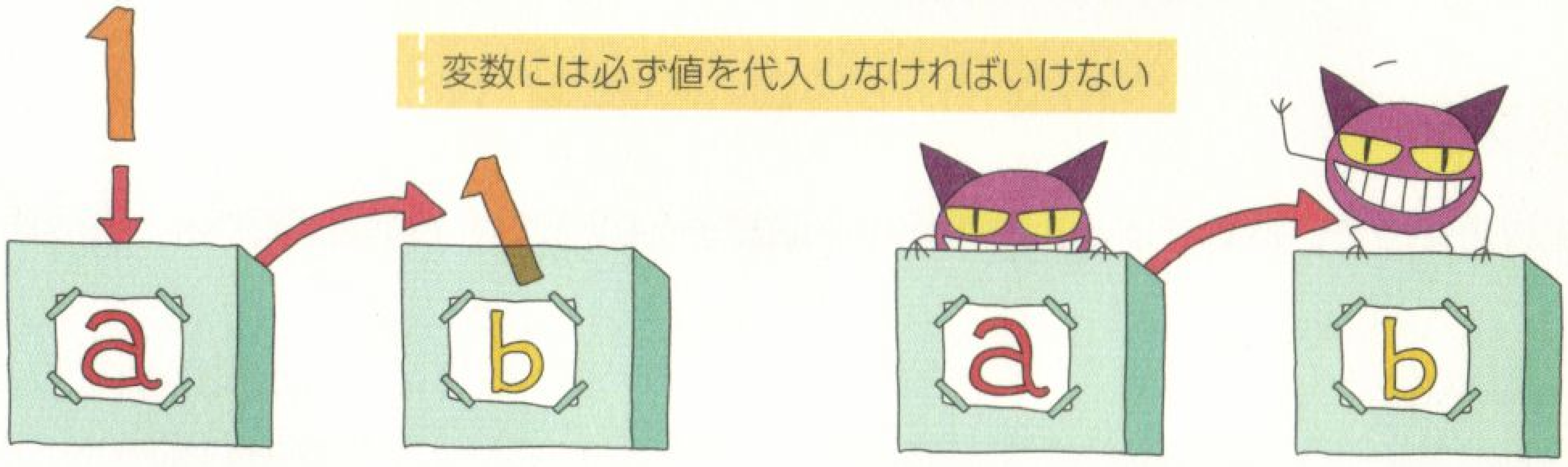
値を代入しないと変数は使えない
（本文78ページ 変数の使い方のまとめ参照）



▶専門的な説明

なぜ、変数に値を代入しないと使えないかというと、変数は、データの一時的な保管場所を、コンピュータの記憶領域（メモリ）に確保します。何も代入しないと、ここに何が入っているかわかりません。もしかすると、コンピュータの動作をおかしくするような値が入っているかもしれません（これを応用したウイルスも存在します）。

そのために、変数には、必ず値を代入しないといけないのです。



COLUMN 数当てゲーム その2（本文66ページのつづき）

本文66ページでは、数当てゲームをしました。ここでも、数当てゲームをしてみましょう。でも、今度の当て方は、当たり前のことです。

0、1、2、3、4、5、6、7の数字を、「0から3」と、「4から7」との2つのグループに分けます。はじめに、相手の思っている数字が、どちらのグループにあるのかを答えてもらいます。

そして、「ある」と答えたグループを2つに分けます。例えば、「0から3」のグループに、「ある」と答えたのならば、「0、1」と「2、3」に分けます。

次に、相手の思っている数字が、どちらのグループにあるのかを答えてもらいます。それをまた、2つのグループに分けてもらいます。「0、1」のグループならば、「0」、「1」です。

最後に、このどちらかを相手に聞きます。

当たり前ですが、相手の思っている数字が当たりますね。

Section 02

計算するための記号 ——演算子

演算子とは、計算するための記号のことです。算数や数学で使ったことのある、「=」、「−」、「+」なども演算子です。ちょっと注意が必要なのは、「=」は「代入」であるということです。ここでは、C言語の演算子の使い方を見ていきましょう。

演算子

変数を使って、計算をするための記号を**演算子**といいます（付録206ページ参照）。

▼基本的な演算子

演算子	算術演算子	+ - ＊ / % ++ --
	関係演算子	< <= > >= == !=
	論理演算子	&& || !
	代入演算子	=

ここでは、算数や数学で馴染み深い、足し算や引き算などをするための**算術演算子**を使ったソースコードを見てみましょう。先ほどのソースコード「MyC3.c」を、算術演算子を使って、ちょっと変えてみました。

```
/* MyC3.c */
#include <stdio.h>

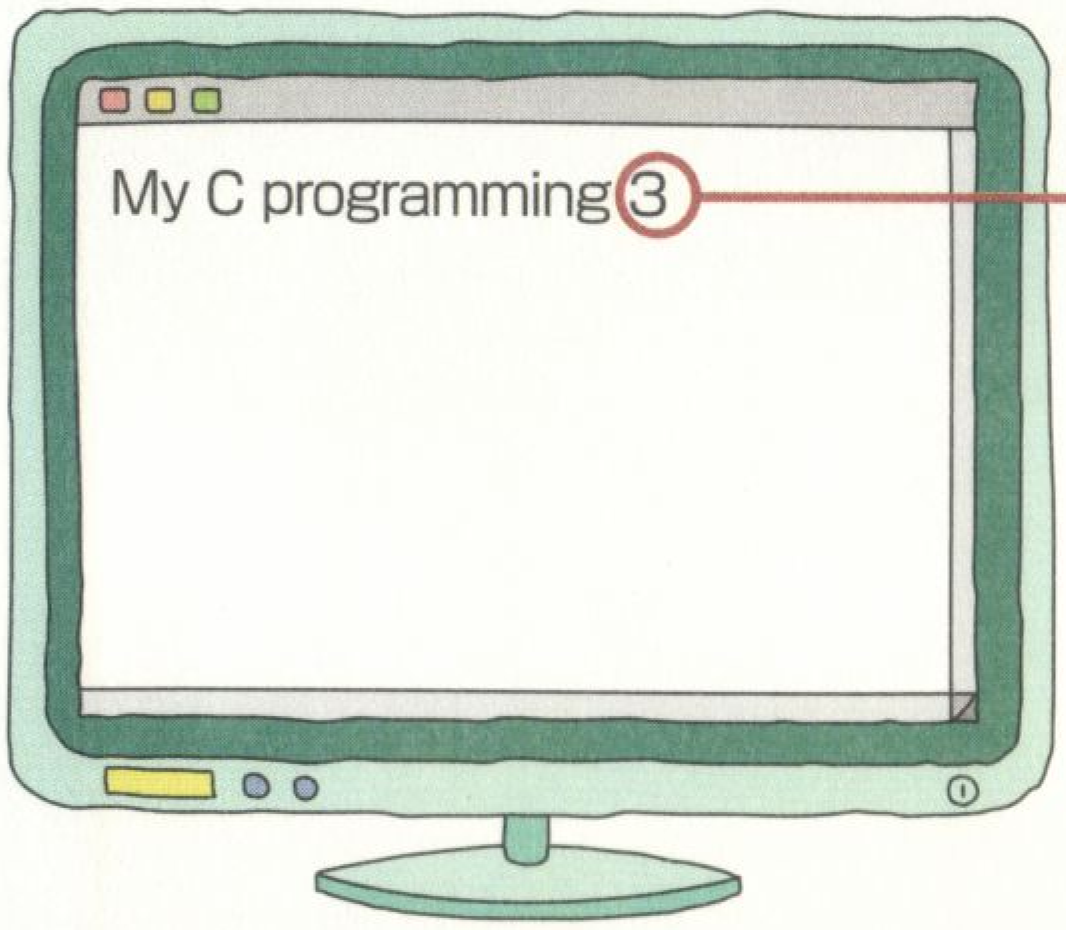
int main(void)
{
  int number;

  number = 2 + 1;
  printf("My C Programming%d¥n",number);
  return 0;
}
```

→ 変数numberに、「2+1」の値を代入

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC3.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「`cd c:¥MyC` Enter」と入力してください。
❺「`gcc MyC3.c -o Myc3` Enter」と入力します。
❻「`MyC3` Enter」と入力します。



◀実行結果

算数や数学で、馴染み深い算術演算子の書き方と意味は、次のようになります。
計算の優先順位は、算数や数学と基本的に同じで、式にはカッコ( )を用いることもできます。

▶専門的な説明

式という言葉が出てきましたが、ここで式について簡単に説明してみましょう。

プログラミング言語でいうところの式とは、数式はもちろんですが、値を返す文だと考えてください。そのため、式には必ずその結果を出力する出力先（例えば、変数）が必要です。

▼算術演算子①

演算子	書き方	意味
+（プラス）	A + B	AとBを足す
-（マイナス）	A − B	AからBを引く
*（アスタリスク）	A * B	AとBを掛ける ➡ 算数や数学での「A × B」
/（スラッシュ）	A / B	AをBで割る ➡ 算数や数学での「A ÷ B」
%（パーセント）	A%B	AをBで割った余りを求める ➡「A ÷ B」の余り

算術演算子には、次のようなものもあります。

▼算術演算子②

演算子	書き方	意味
++	A++	Aを使ってからAの値を1増やす ➡ 「A = A + 1」
--	A--	Aを使ってからAの値を1減らす ➡ 「A = A − 1」

例題 算術演算子❶

次のようなソースコードを入力して実行すると、「My C program□」の「□」の部分はどんな数になりますか。

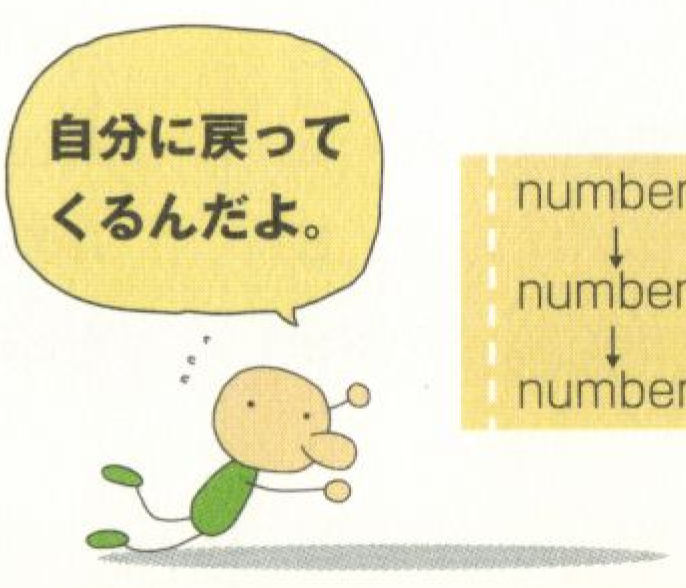
```
 1:/* Exercise4.c */
 2:#include <stdio.h>
 3:
 4:int main(void)
 5:{
 6:  int number;
 7:
 8:  number = 3;
 9:  number = number + 1;
10:  printf("My C Program%d",number);
11:  return 0;
12:}
```

●解答

4

●解説

8行目で、変数numberには、「3」が代入されています。9行目で、この変数numberの値「3」に「1」を足して、再度、変数numberに代入していますから、変数



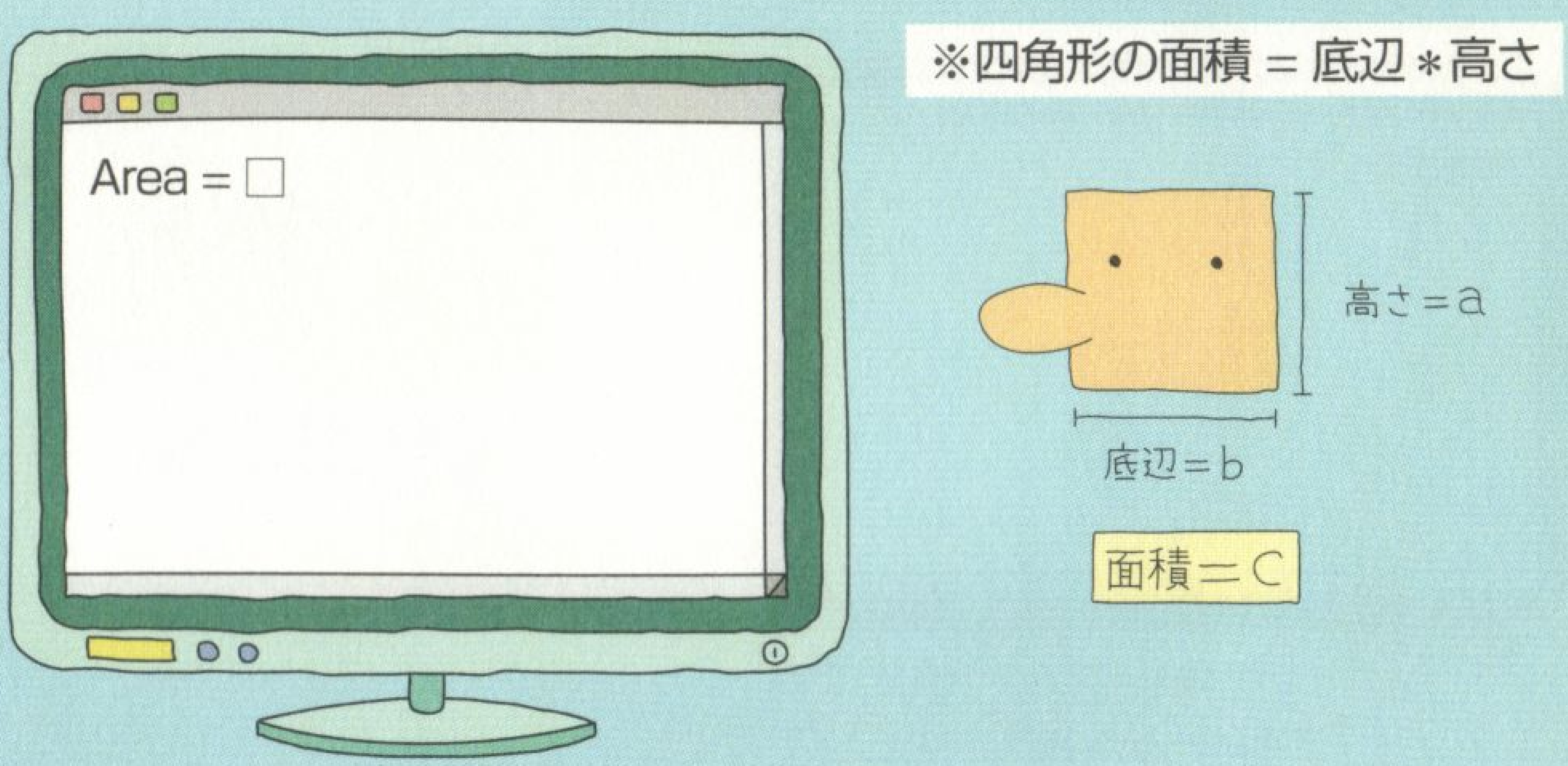
number=3
↓
number=number +1
↓
number=4

numberの値は「4」になります。

算数や数学とは違うので、変に感じるかもしれませんが、コンピュータのプログラミングでは、変数が自分の値を演算した結果を、自分自身に戻すことはよく使われます。

例題 算術演算子❷

❶底辺をint型変数aの値、高さをint型変数bの値とする四角形の面積を、int型変数cとして求め、次のように画面に出力するソースコードを書いてください。



●解答

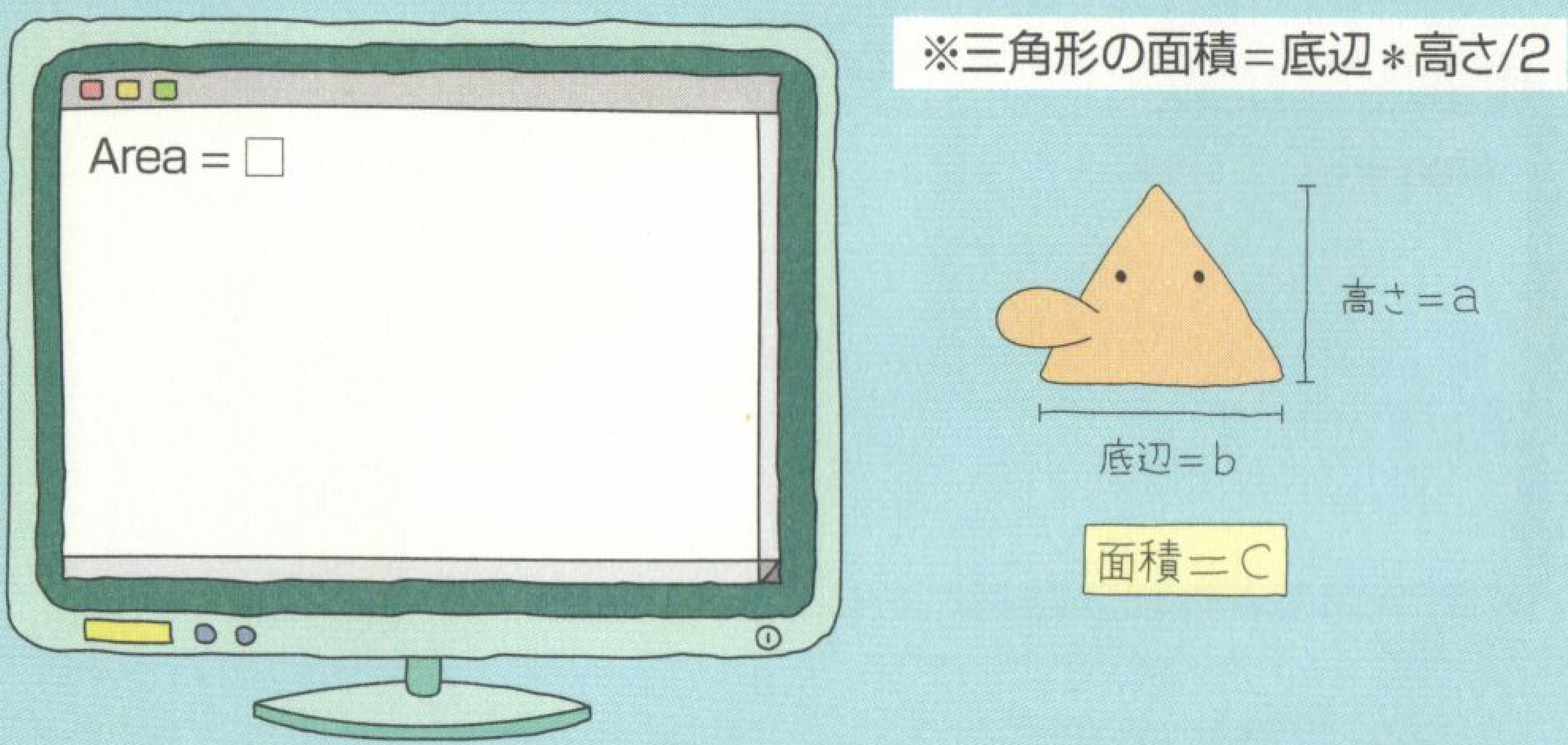
```
/* Exercise5.c */
#include <stdio.h>

int main(void)
{
  int a;
  int b;
  int c;

  a = 5;  → 例として5
  b = 3;  → 例として3
  c = a * b;  → かけ算
```

```
  printf("Area = %d¥n", c);
  return 0;
}
```

❷高さをfloat型変数aの値、底辺をfloat型変数bの値とする三角形の面積を、float型変数cとして求め、次のように、画面に出力するソースコードを書いてください。



●解答

```
/* Exercise6.c */
#include <stdio.h>

int main(void)
{
  float a;
  float b;
  float c;

  a = 5;
  b = 3;
  c = (a * b) / 2;
```

```
13:  printf("Area = %f¥n", c);
14:  return 0;
15:}
```

●解説

12行目を見てください。カッコを付けたところが優先的に計算されるのは、算数や数学と変わりません。

12行目を、例えば、次のようにしてもかまいません。

```
c = (a * b) / 2
```

⬇

```
c = a * b
c = c / 2
```



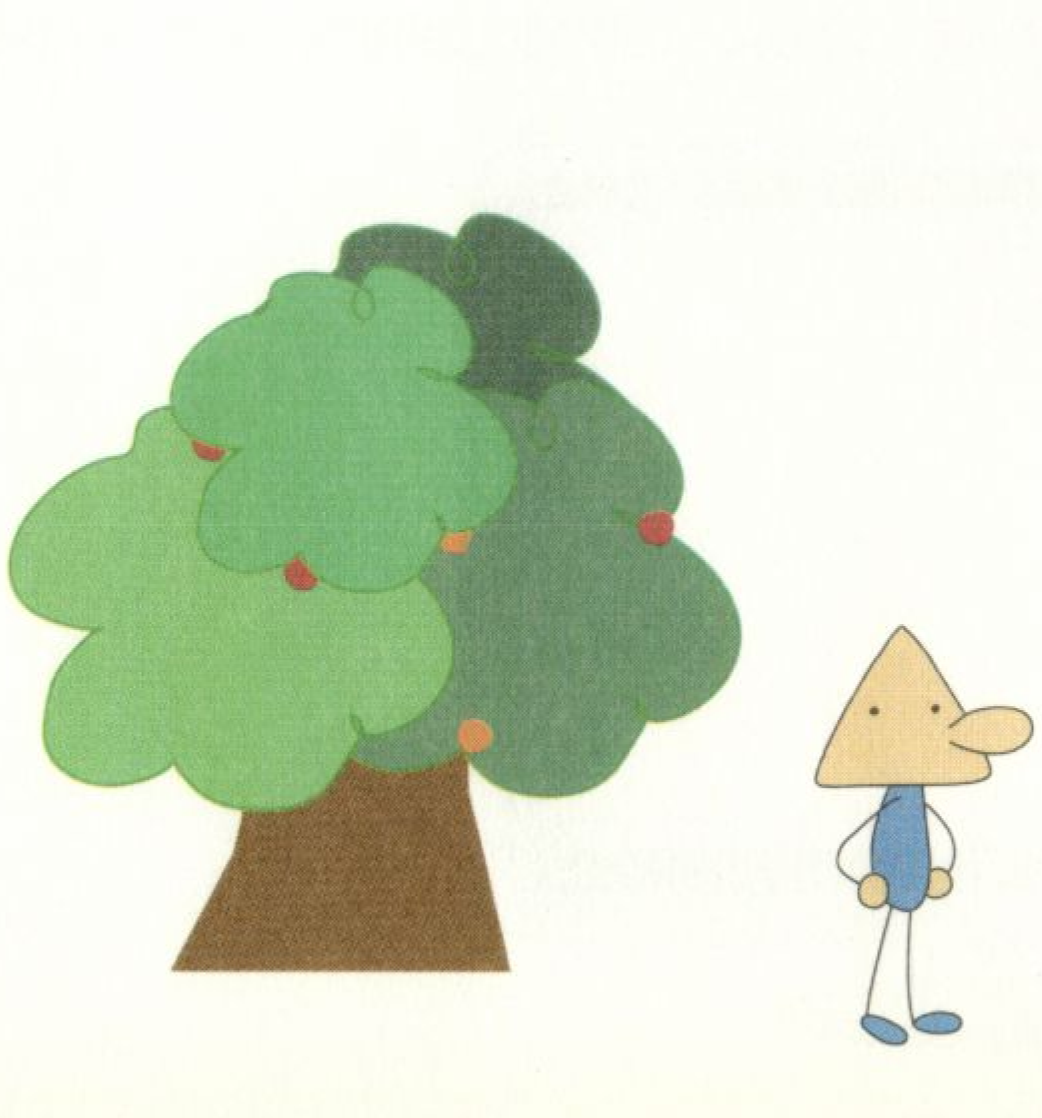
▶専門的な説明

この例題で変数の型をfloat型としたのは、なぜでしょう？

それは、底辺と高さが整数でも、面積を求めるときに、2で割ることで変数cが小数になることがあるからです。

ただ、それなら、変数cだけをfloat型にすればよいかと思うかもしれませんが、自動的に「型変換」というものが行われてしまうので期待した結果が得られません。

変数の型を、いろいろと変化させてみて結果を確認してみましょう。



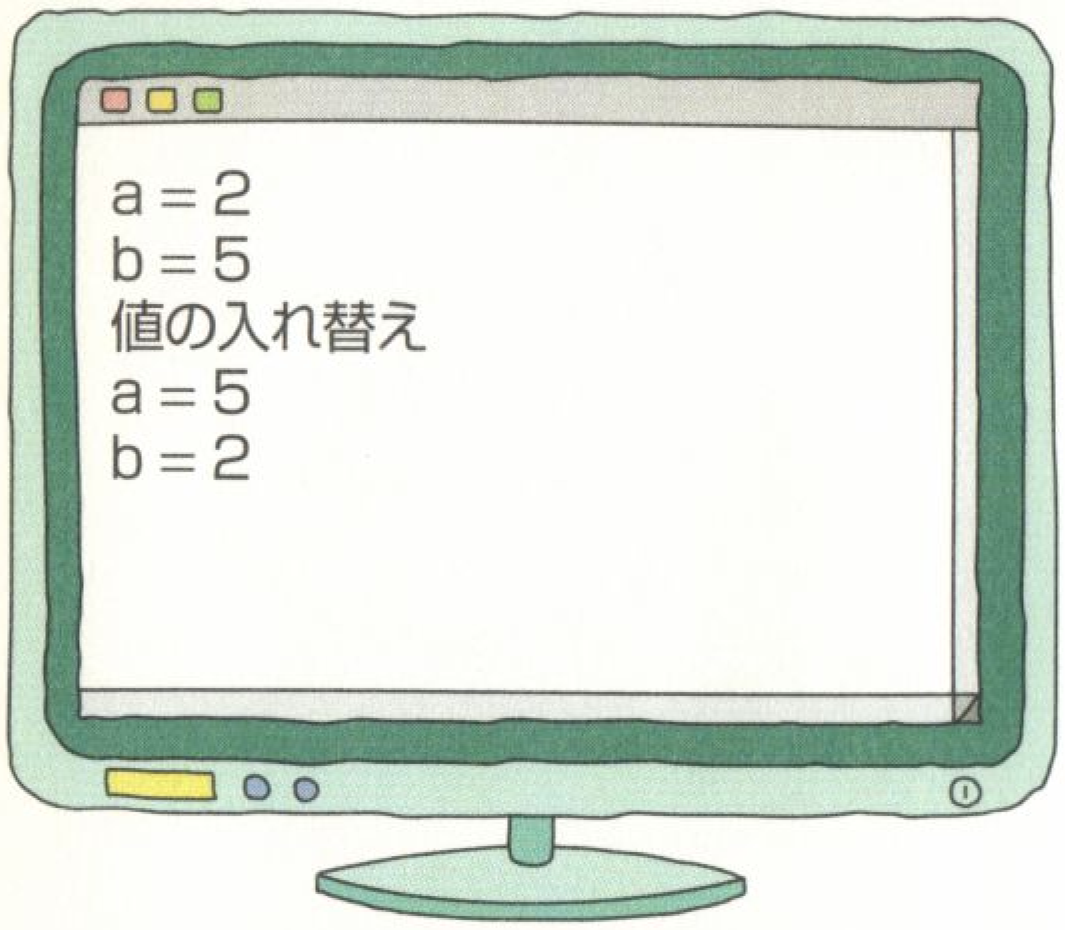
practice 練習問題

解答は213ページ

1 変数　難易度★★★★☆

はじめに、変数aに値2を代入し、変数bに値5を代入して、次に変数ａと変数ｂの値を変数と変数の代入を使って入れ替えて、次のように、画面に出力するソースコードを書いてください。

▶▶ヒント：本文76ページ参照



ソースコードで使う変数は、aとb以外にもう1つ必要です。

2 変数と演算子　難易度★★★☆☆

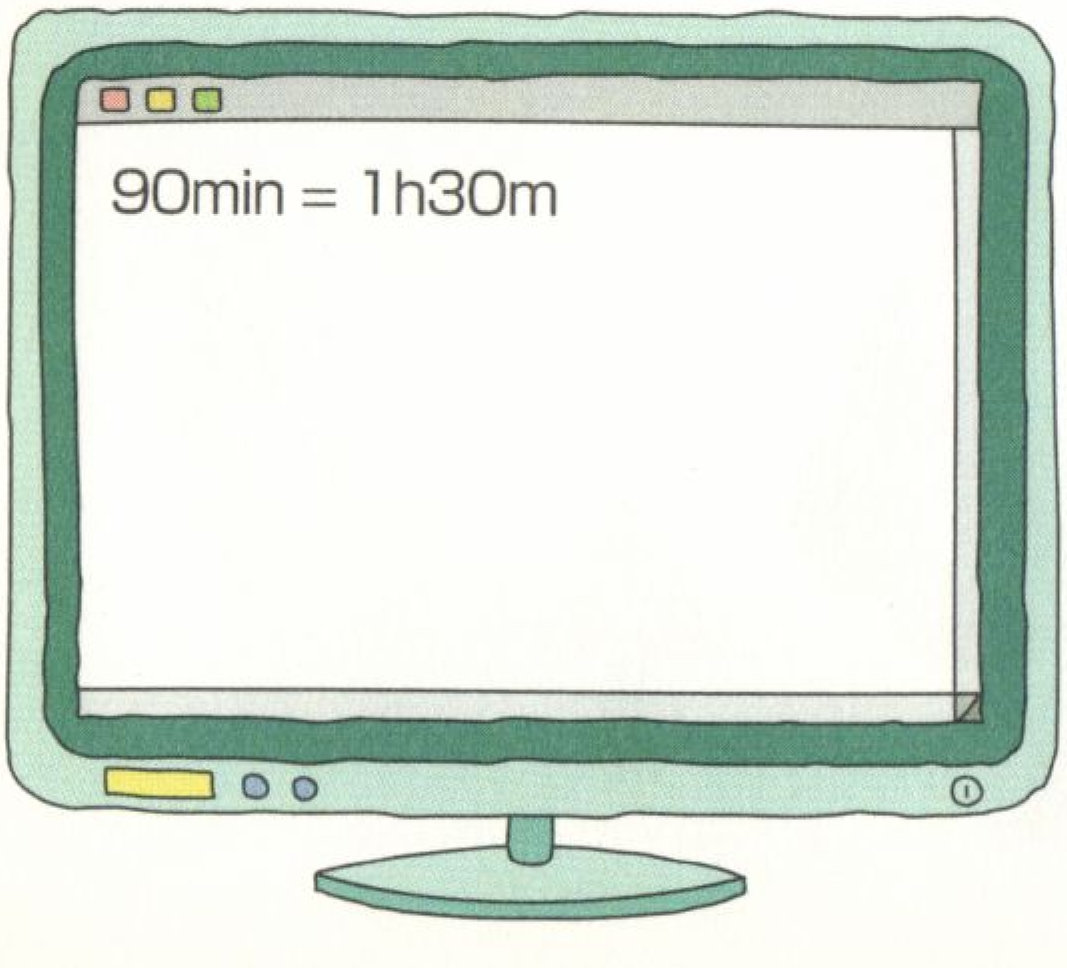
円の半径をdouble型の変数ｒ、円周率を3.141592654として変数piに代入して、その円の円周と面積を出力するプログラムを書いてください。

▶▶ヒント：本文84ページ参照

3 変数と演算子　難易度★★★★☆

int型の変数aを使って、次のように画面に出力するソースコードを書いてください。

▶▶ヒント：1時間は60分



Chapter4のまとめ

★**変数**

変数とは、データを記憶するものです。

★**変数の名前**

複数の変数を区別するのに、変数には名前を付けなければなりません。

★**識別子**

識別子とは、変数やメソッドなどに付けられた名前のことです。

★**データ型（変数の型）**

変数はデータの種類によって型が決まります。

★**変数の宣言と初期化**

変数を使うには、宣言（変数を使う前に、ソースコードで、名前とデータ型を指定すること）と初期化（宣言したあと、はじめてデータを代入すること）が必要です。

★**代入**

代入とは、変数にデータを記憶させることで、とても重要な考え方です。

★**演算子**

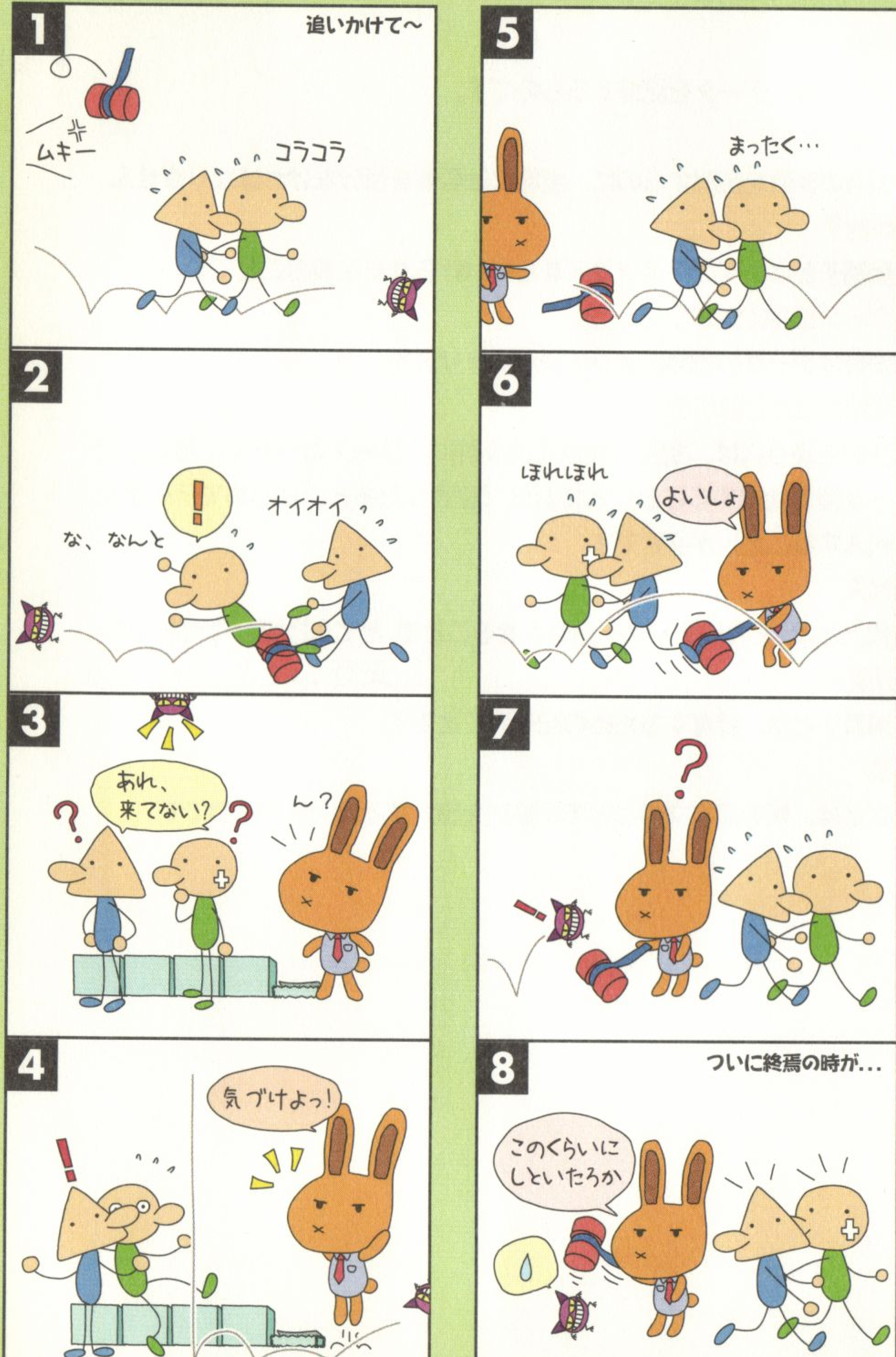
演算子とは、計算するための記号のことです。

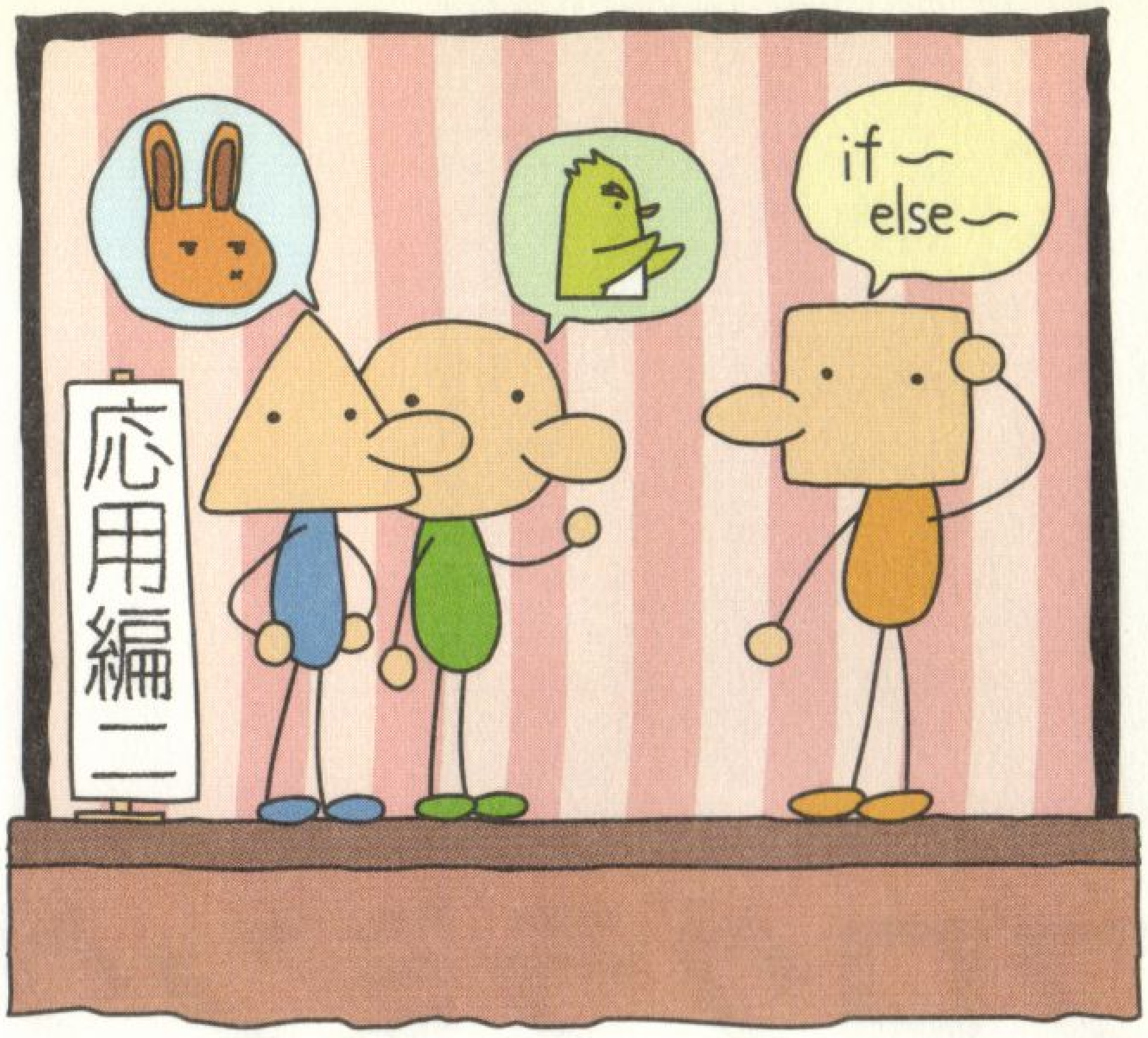
★**式**

式とは、値を返す文のことで、結果を出力する出力先が必要です。

ドキドキ4コマ

バグくんをつかまえろ②





★第4章に続いて、C言語のプログラミングをします。
プログラムにいろいろな制限を付け、流れを変えたり、

工夫をしてみましょう。

★C言語のより高度なプログラミングを学びたい方は

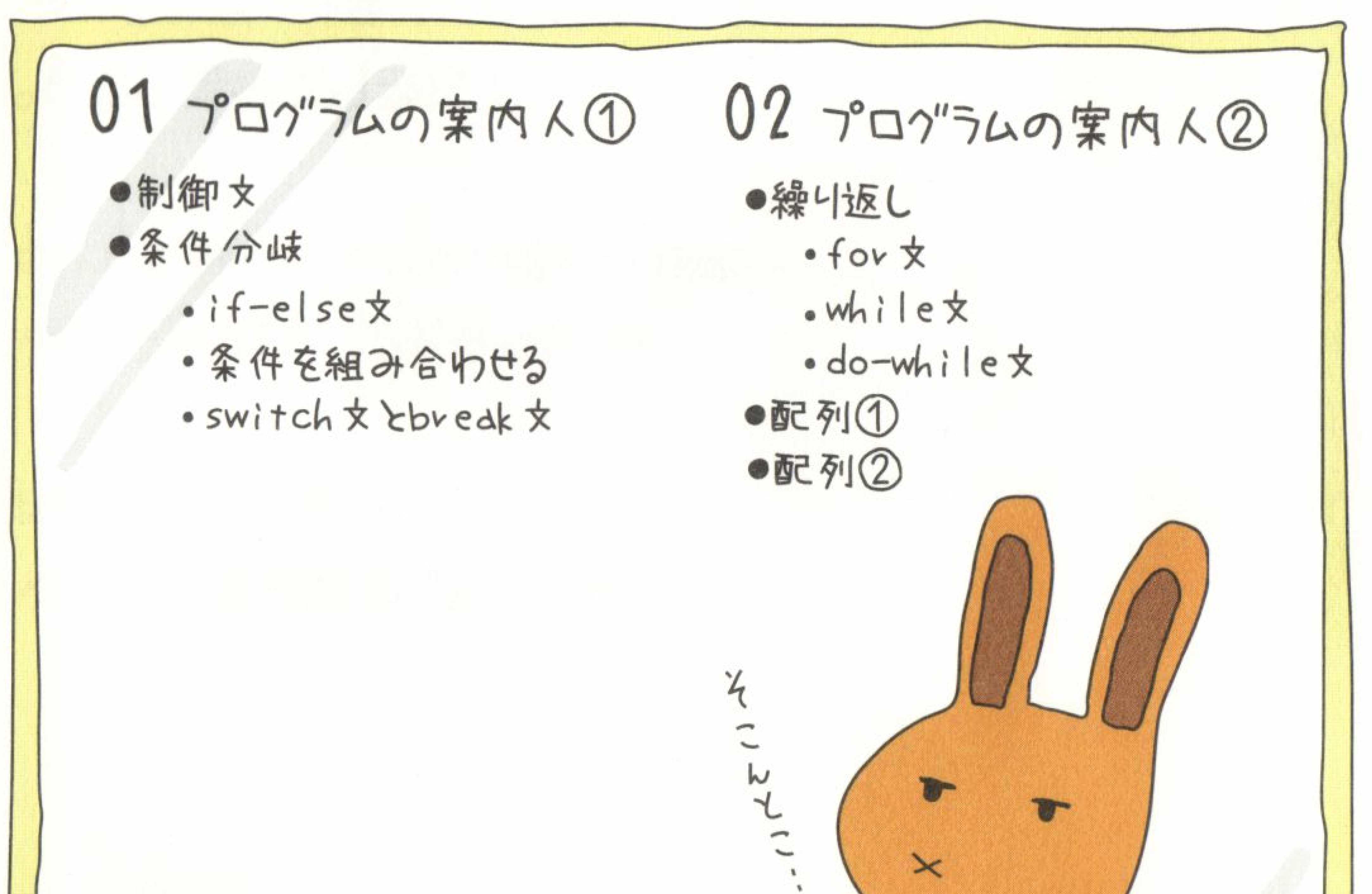
➡第6章本文138ページへ

★C言語の基本を確認したい方は

➡第3章本文54ページへ

Chapter 5

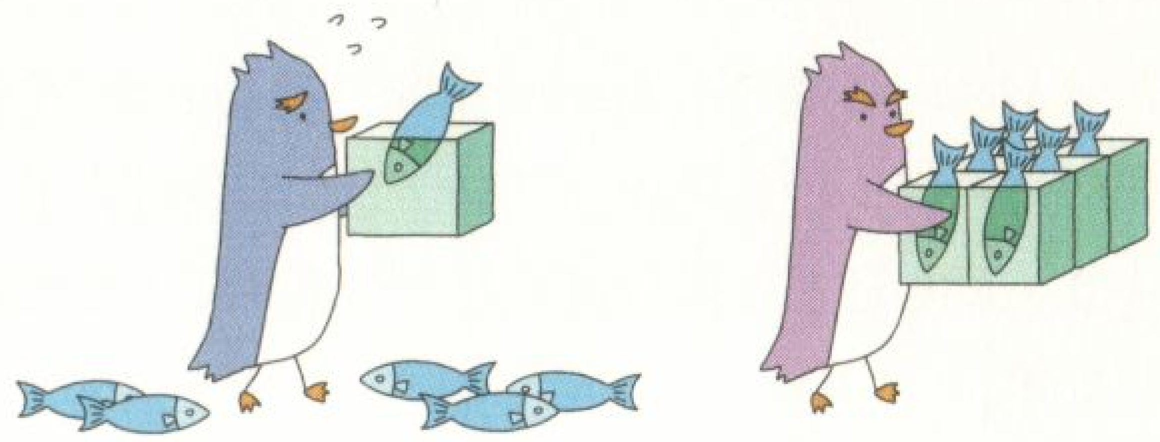
プログラミングの心得❷



この章でマスターすること

コンピュータにより多くのことをしてもらうためには、様々な場合やデータに対応できるような文の書き方ができればよいですね。それは、プログラミング言語で、ものごとを表現する場合に、そのものごとに合わせて、コンピュータに指示する動作を柔軟に変更できるということでもあります。

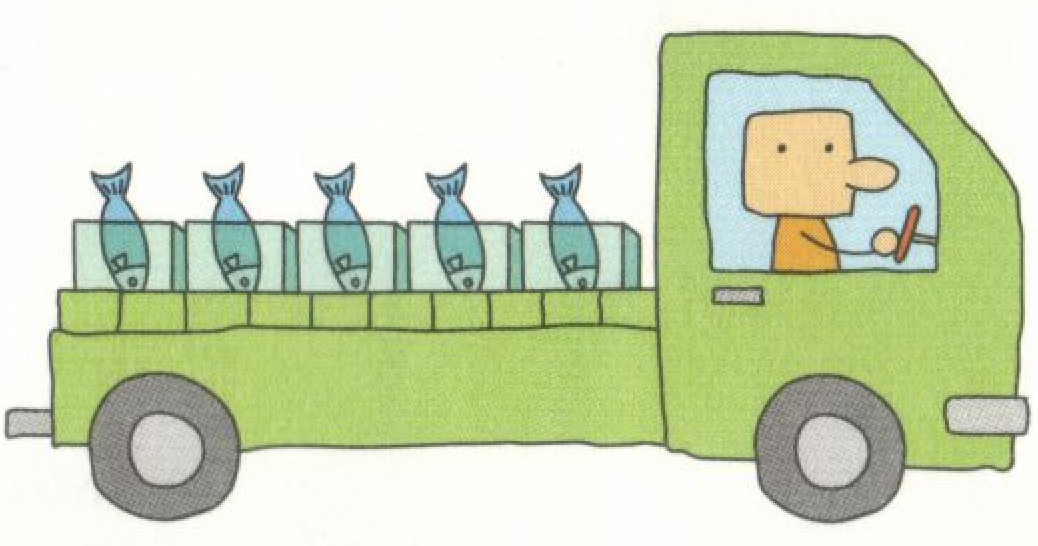
そのために、C言語には、**制御文**というものがあります。制御文には、**条件分岐**と**繰り返し**があります。

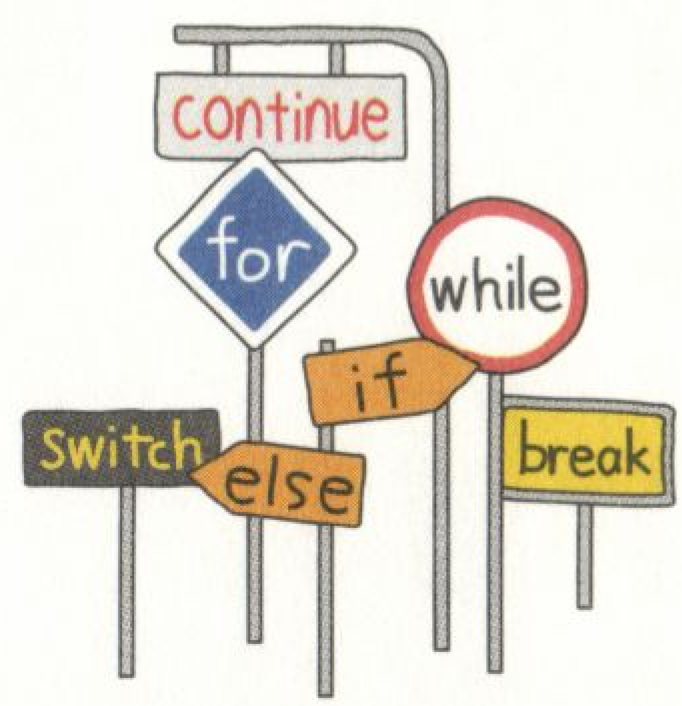


また、繰り返しの操作といっしょに使うと便利な、**配列**というデータの扱い方もあります。

この章では、制御文の基礎を説明していきます。

制御文ってどんな種類があるんだろう?





C言語必須用語

制御文　**条件分岐**
関係演算子　**論理演算子**
繰り返し　**配列**

ステップアップ用語

制御構造
繰り返しと配列

Section 01

プログラムの案内人❶

C言語のプログラムは、他のプログラミング言語と同じように、ふつうは上から下へと文が実行されていきますが、その実行の流れを変える「制御文」というものがあります。ここでは、制御文とは何であるかを説明したあと、制御文の一つである「条件分岐」について説明します。

制御文

プログラムは、基本的に「main関数」の上から下に向かって、ソースコードに書かれた順番に実行されていきますが、プログラムの内容によっては、順番に実行したくないこともあります。同じことを繰り返すときや条件によって違ったことをさせたい場合が、これにあたります。

どういうことかを簡単に説明してみましょう。

例えば、ある書類のコピーをAさんからCさんまで、3人ぶん必要だとします。コピー機は、コピー室にあります。コピーを3人ぶん取ることを、ソースコードふうに書いてみます。

```
 1://コピーを取ることのソースコードふう
 2: コピー室へ行く
 3: コピーを取る（Aさんのぶん）
 4: 席へ戻る
 5: コピー室へ行く
 6: コピーを取る（Bさんのぶん）
 7: 席へ戻る
 8: コピー室へ行く
 9: コピーを取る（Cさんのぶん）
10:席へ戻る
```



2行目から4行目までの内容を、3回繰り返して書かなければなりません。もし、これが3人ぶんではなくて、100人ぶんだったらどうでしょうか。気が遠くなりますね。

また、コピー室へ行くことができなくなっていたら、コピーを取ることは無理です。

C言語には**制御文**というものがあり、これを使えば、条件によって実行せずに飛ばしたり、同じ処理を繰り返したりといったことができるようになります。

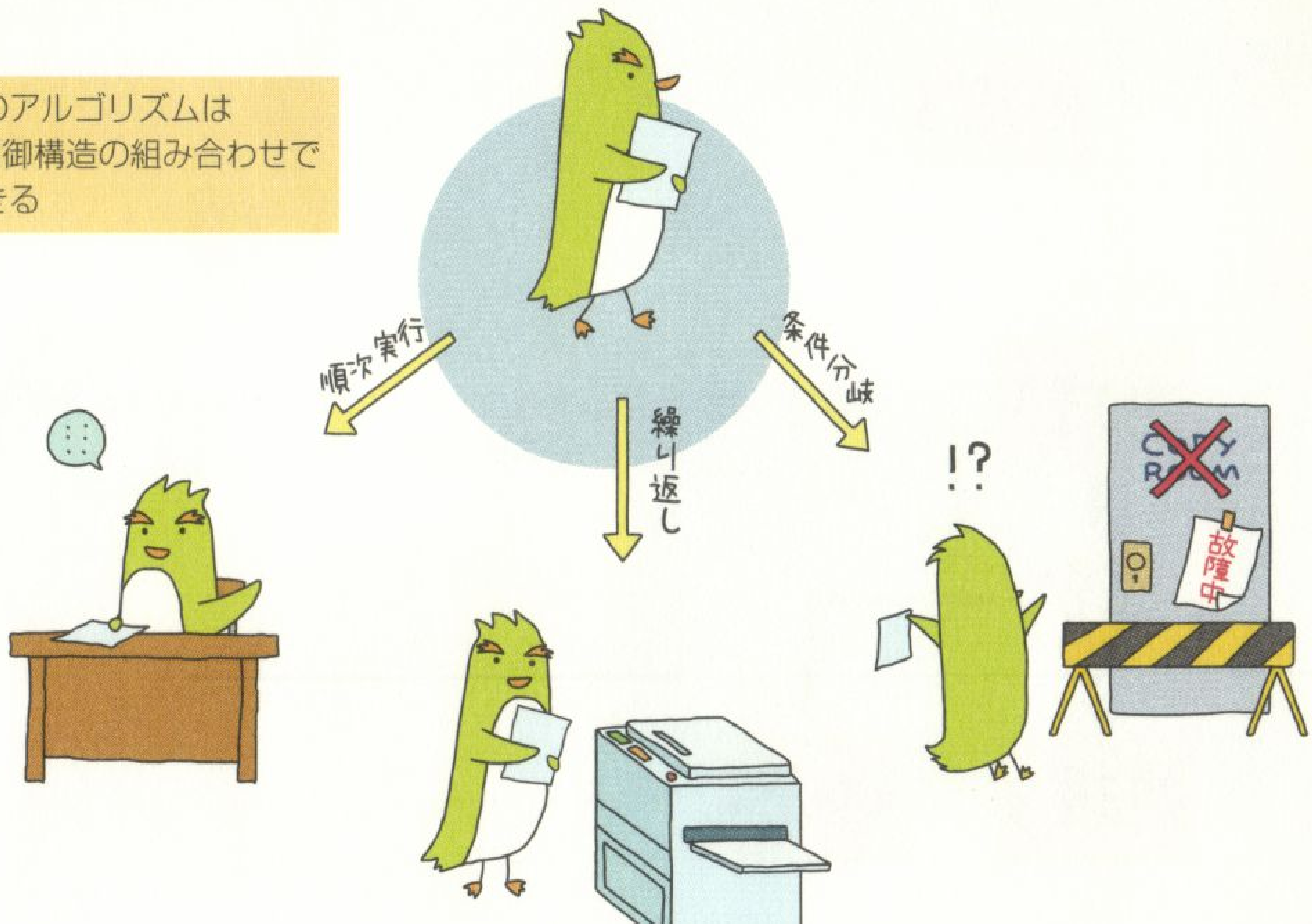
制御文には**条件分岐**と**繰り返し**があります。

▶専門的な説明

プログラムは、基本的に実行文を上から順に実行していく単純な構造（順次実行）をしていますが、実際のプログラムでは、条件によって実行する文を変更したり、同じ文を何回も実行したりと、必ずしも上から順に実行するわけではありません。このようなプログラムの流れを決定するものを、プログラムの**制御構造**といいます。

また、とても大事なことなのですが、すべてのアルゴリズムは、たった3つの制御構造の組み合わせで記述することができるのです。その3つの制御構造とは、**順次実行**、**分岐実行**（**条件分岐**）、**繰り返し実行**（**繰り返し**）です。

すべてのアルゴリズムは
3つの制御構造の組み合わせで
記述できる



COLUMN 数当てゲームの仕組み－情報量（本文83ページのつづき）

本文66ページと83ページで、数当てゲームをしました。

コンピュータの発展には、情報の量（情報量）を、数として測れることが不可欠だったのですが、この数当てゲームの仕組みは情報量を表していたのです。

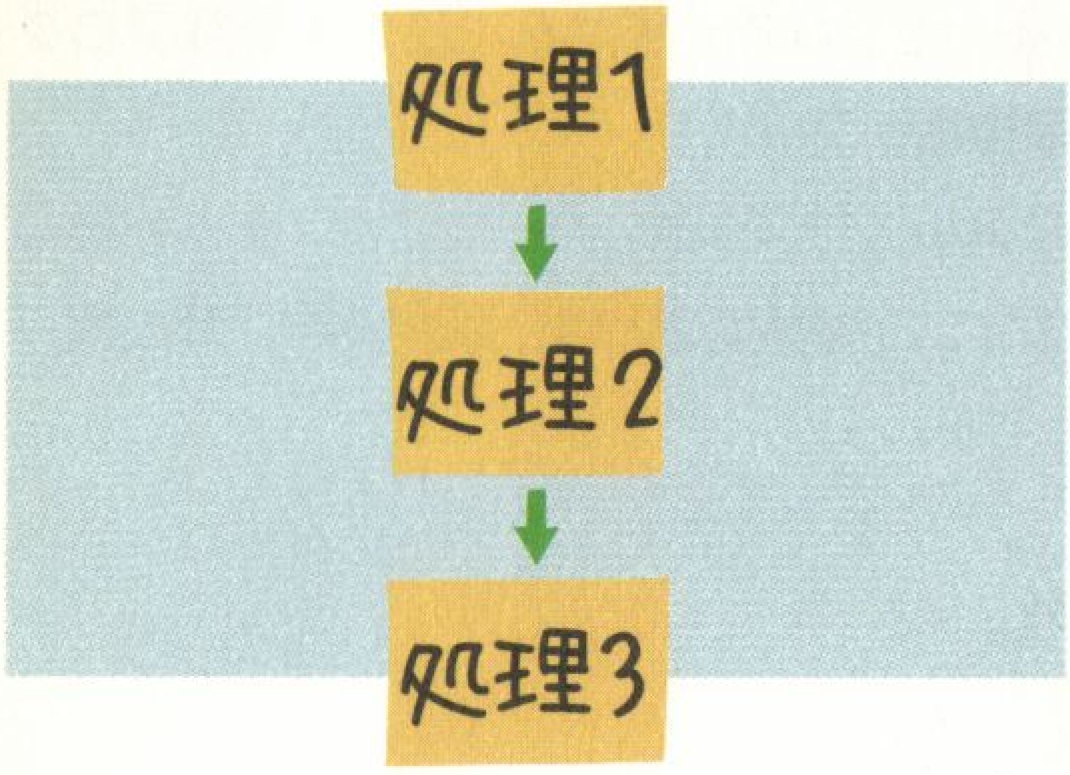
上記の2つのコラムも、相手に対する質問の数は3回でした。つまり、8つの数から、相手の思っている1つの数を当てるには、どうしても3回の質問が必要だということです。

これを言い換えると、「8つの数から、相手の思っている1つの数を当てるには、情報量が3必要になる」ということになります。

コンピュータの世界でよく使われる、「ビット」という単位は、この情報量を表しています。1ビットとは情報量1のことをいいます。

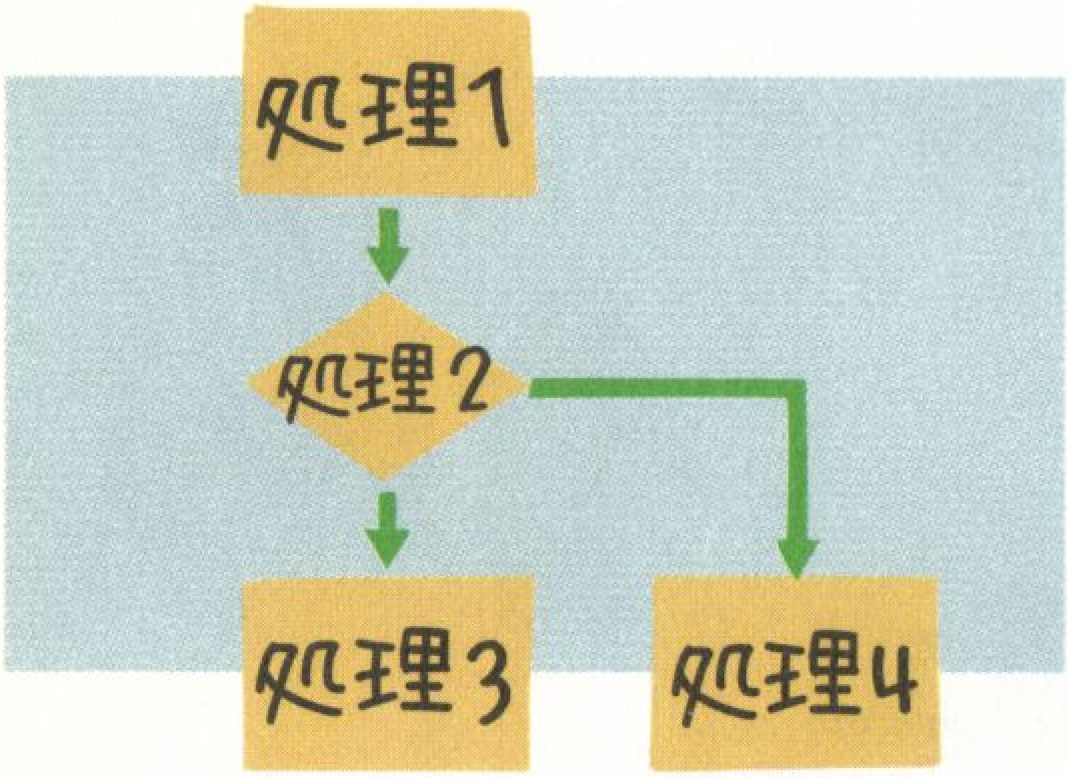
図で、3つの制御構造をもう一度確認してみましょう。

▼順次実行



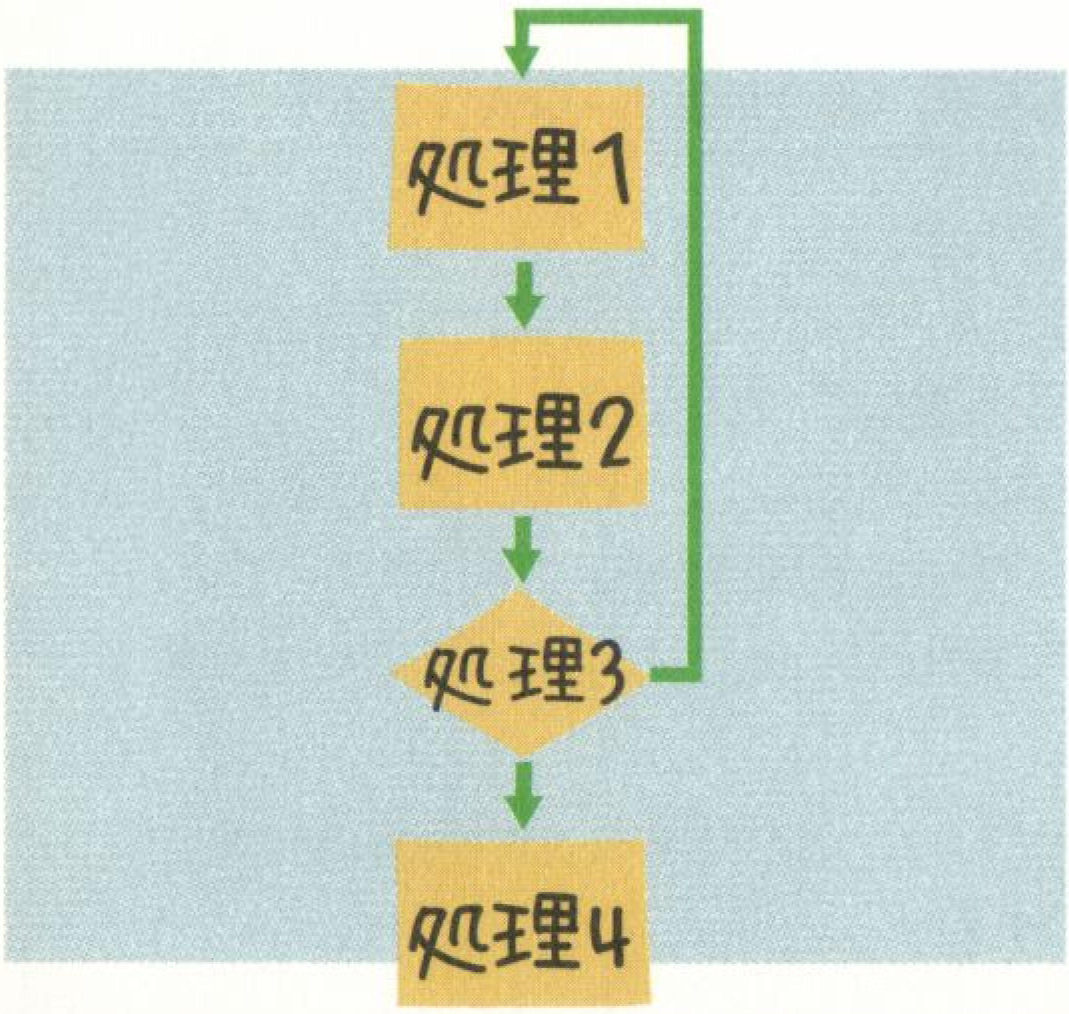
プログラムを上から順に実行していくことを**順次実行**といいます。

▼分岐実行（条件分岐）



条件によって、プログラムの流れを変更することを**条件分岐**といいます。

▼繰り返し実行（繰り返し）

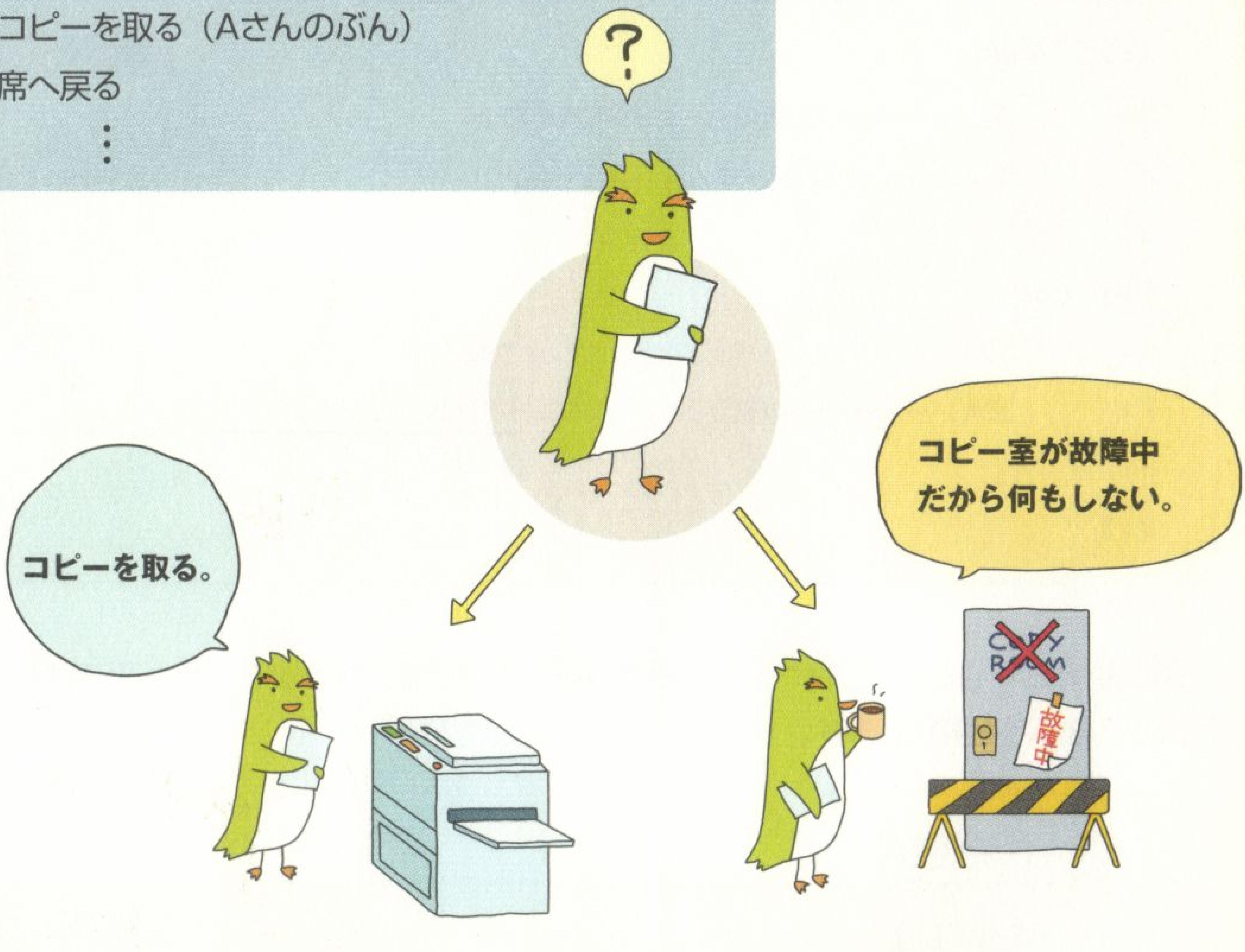


条件が成立するまでプログラムを繰り返すことを**繰り返し**といいます。

条件分岐

ある条件が成立するか、しないかによって、実行する文やブロックを選択できるようにするのが**条件分岐**です。先ほどの例で考えると、コピー室へ行くことができなくなっていたときなどに使えばよいでしょう。

```
//条件分岐を使ってみる
「もし、コピー室へ行くことができないならば」
何もしない
「そうでなければ」
コピー室へ行く
コピーを取る（Aさんのぶん）
席へ戻る
        ⋮
```



C言語には、条件分岐に**if-else文**、**switch文**、**break文**があります。

▶▶if-else文

if-else文とは、ifが「もし～ならば」を表し、elseは「そうでなければ」を表します。また、if-else文には次のように3つの書き方があります。

▶if文

書式

```
1:if (条件) {
2:      文1; → 条件が成立したら実行する文
3:      文2; → 条件が成立したら実行する文
4:       ⋮
5:}
```

条件が成立すると2行目から4行目を実行しますが、そうでなければ5行目に進みます。

▶if-else文

書式

```
1:if (条件) {
2:      文1; → 条件が成立したら実行する文
3:      文2: → 条件が成立したら実行する文
4:       ⋮
5:} else {
6:      文3; → 条件が成立しなかったら実行する文
7:      文4; → 条件が成立しなかったら実行する文
8:       ⋮
9:}
```

条件が成立すると、2行目から4行目を実行して9行目に進みます。そうでなければ、6行目から8行目を実行します。

▶if-else if-else文

書式

```
1:if (条件1) {
2:      文1; → 条件1が成立したら実行する文
3:      文2; → 条件1が成立したら実行する文
4:       ⋮
5:} else if (条件2) {
6:      文3; → 条件1が成立せず、条件2が成立したら実行する文
7:      文4; → 条件1が成立せず、条件2が成立したら実行する文
```

```
 8:        :
 9:} else {
10:       文5;──→どの条件も成立しなかったら実行する文
11:       文6;──→どの条件も成立しなかったら実行する文
12:        :
13:}
```

条件1が成立すると、2行目から4行目までを実行し、13行目に進みます。そうでなくて、条件2が成立すると、6行目から8行目までを実行し、13行目に進みます。どの条件も成立しないと、10行目から12行目を実行します。

それでは、実際にC言語で「if-else文」を使ってみましょう。

```
/* MyC4.c */
#include <stdio.h>

int main(void)
{
  int a;
  a = 1;

  printf("a = %d¥n", a);

  if (a < 5){  ──→if文
         printf("Statement 1¥n");
  }

  if (a > 5){  ──→if-else文
         printf("Statement 2¥n");
  } else {
         printf("Statement 3¥n");
  }
```

```

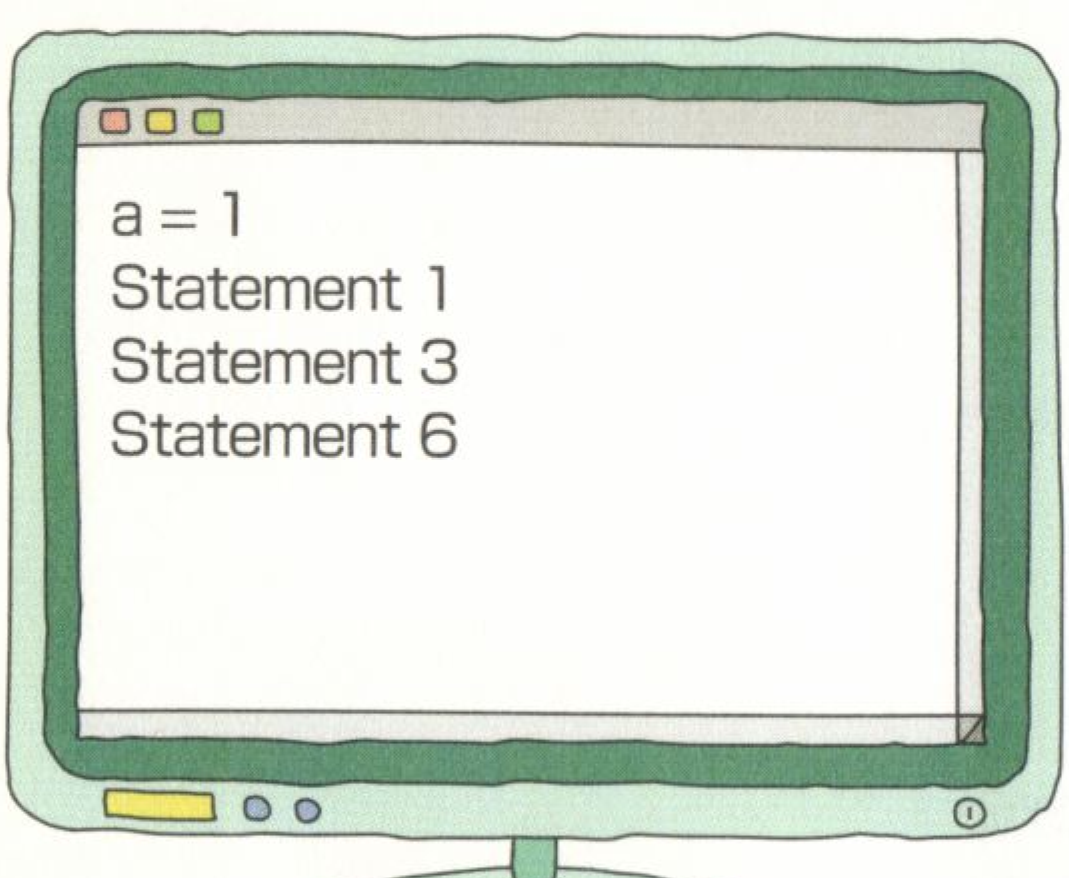
    if (a > 5){
            printf("Statement 4¥n");
    } else if (a < 0){
            printf("Statement 5¥n");
    } else {
            printf("Statement 6¥n");
    }

    return 0;
}
```

→ if-else if-else文

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷ 「MyC」フォルダにファイル名「MyC4.c」で保存します。
❸ コマンドプロンプトを起動します。
❹ 「**cd c:¥MyC** Enter」と入力してください。
❺ 「**gcc MyC4.c -o MyC4** Enter」と入力します。
❻ 「**MyC4** Enter」と入力します。





●解説

11行目から13行目で、**if文**を使っています。

ここの意味は、「もし変数aの値が5より小さければ、『statement 1』と画面に出力する」です。変数aには、7行目で「1」を代入していますので、画面には「statement 1」と表示されます。

15行目から19行目で、**if-else文**を使っています。

ここの意味は、「もし変数aの値が5より大きければ、『statement 2』と、そうでなければ、『statement 3』と画面に出力する」です。変数aには、7行目で「1」を代入していますので、画面には「statement 3」と表示されます。

21行目から27行目で、**if-else if-else文**を使っています。

ここの意味は、「もし変数aの値が5より大きければ、『statement 4』、そうではなくて、0より小さければ『statement 5』、それらのどちらでもなければ『statement 6』と画面に出力する」です。変数aには、7行目で「1」を代入していますので、画面には「statement 6」と表示されます。

▶▶条件を組み合わせる

サンプルソースコード「MyC4.c」の11行目の「a < 5」という部分は、aは5より小さいという**条件**を表しています。C言語で条件を書くためには、次のような**関係演算子**や**論理演算子**を使います。

▼関係演算子

演算子	書き方	意味
>	A > B	AがBより大きいとき成立
>=	A >= B	AがB以上のとき成立
<	A < B	AがB未満のとき成立
<=	A <= B	AがB以下のとき成立
==	A == B	AとBが等しいとき成立
!=	A != B	AとBが等しくないとき成立

この行では、「aが5未満」であれば、12行目を実行するということです。

ここで、条件についてもう少し詳しく見てみましょう。

プログラムを作り始めると、変数の大きさを限定したくなるときがあります。

例えば、価格が100万円以上で200万円以下の中古車を探したいというようなときです。

このようなときは、「中古車の価格が100万円以上」という条件と、「中古車の価格が

200万円以下」という条件を組み合わせればいいのです。また、価格が100万円以上で軽自動車の中古車を探したいのならば、「中古車の軽自動車」という条件と「中古車の価格が100万円以上」という条件を組み合わせればよいでしょう。



▼論理演算子

演算子	書き方	意味
&&	A && B	Aという条件が成立し、 かつ Bという条件も成立する ならば
||	A || B	Aという条件が成立するか、 または Bという条件が成立するか ならば
!	!A	Aという条件が成立しないとき

論理演算子を使って先ほどの条件を書いてみると、次のようになります。

「中古車の価格が100万円以上」という条件
「中古車の価格が200万円以下」という条件
→ 組み合わせ

▼

「中古車の価格 >= 100万円」 && 「中古車の価格 <= 200万円」

「中古車は軽自動車」という条件
「中古車の価格が100万円以上」という条件
→ 組み合わせ

▼

「中古車 == 軽自動車」 && 「中古車の価格 >= 100万円」

それでは、実際にC言語で「条件の組み合わせ」を使ってみましょう。

```
/* MyC5.c */
#include <stdio.h>

int main(void)
{
  int a;
  int b;
  a = 5;
  b = 1;

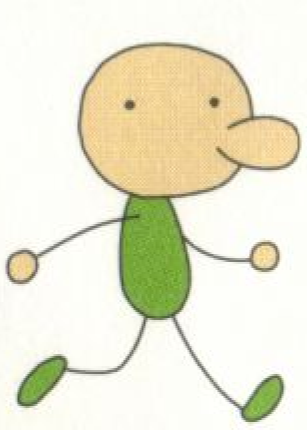
  printf("a = %d¥n", a);
  printf("b = %d¥n", b);

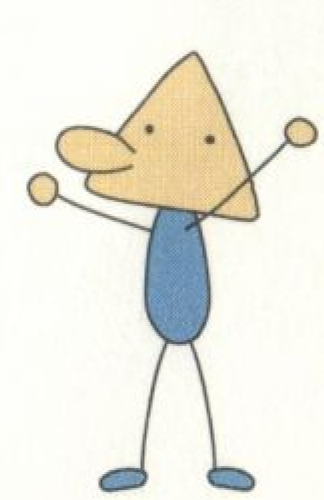
  if (a > 0 && a < 10){ ——→ aが0より大きいかつaが10より小さい
        printf("a > 0 and a < 10¥n");
  }

  if (a > 0 && b == 1){ ——→ aが0より大きいかつbが1である
        printf("a > 0 && b == 1¥n");
  }

  if (a != 0 || b == 1){ ——→ aが0でないまたはbが1である
        printf("a != 0 or b == 1¥n");
  }

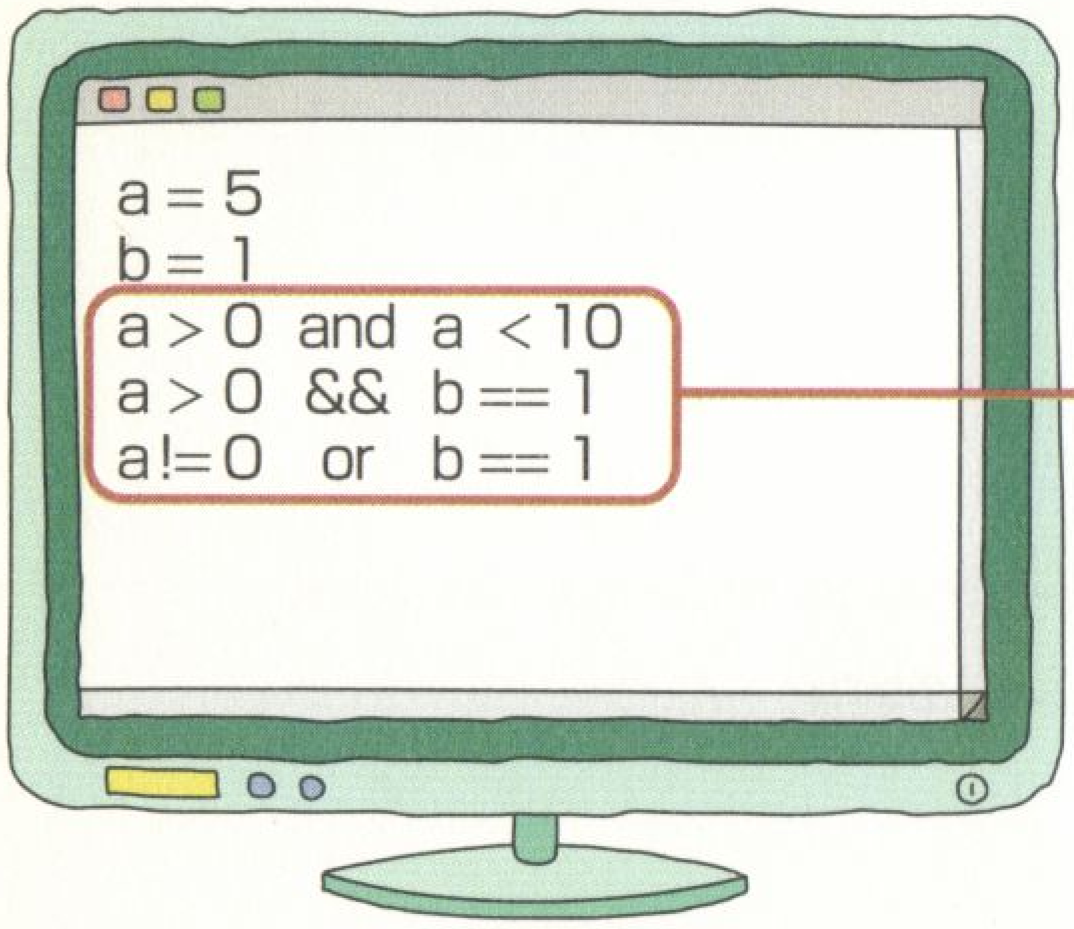
  return 0;
}
```





▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC5.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「`cd c:¥MyC` Enter」と入力してください。
❺「`gcc MyC5.c -o MyC5` Enter」と入力します。
❻「`MyC5` Enter」と入力します。

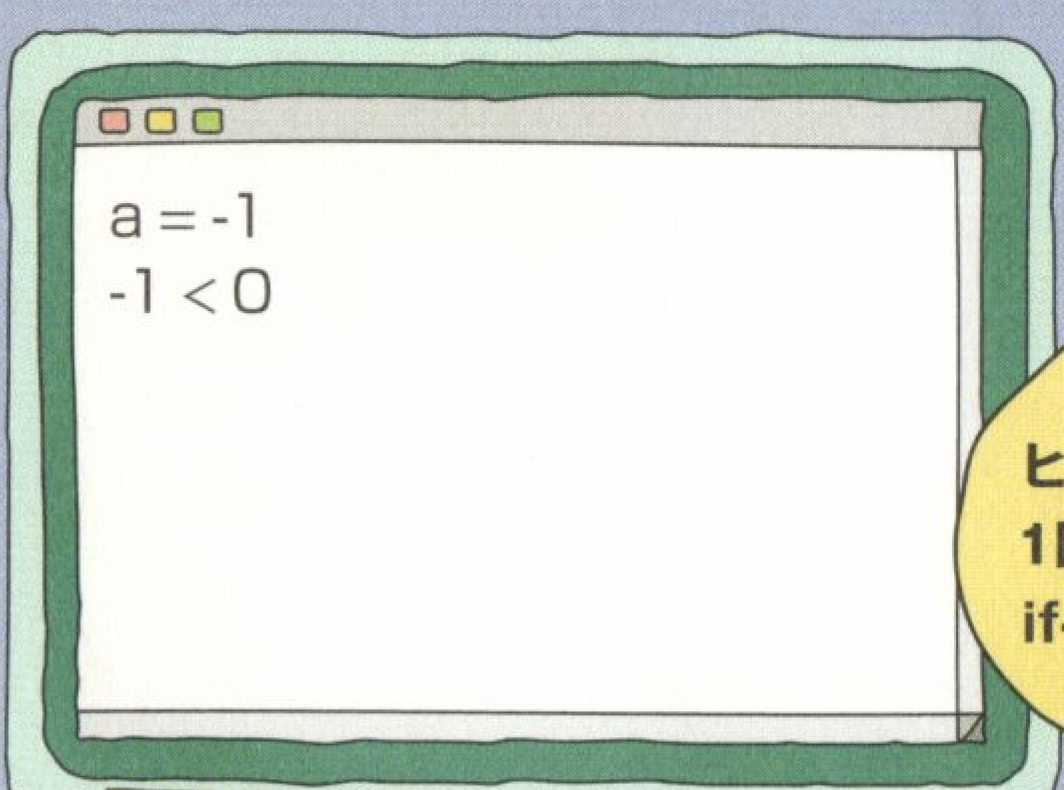


それぞれの条件に合ったものが表示される

例題　条件分岐❶if-else文

int型の変数aに値を代入して、その値が0以上ならば、「□ > =0」、そうでなければ、「□ < 0」というメッセージを出力するソースコードを書いてください。

▼出力例





●解答

```c
/* Exercise7.c */
#include <stdio.h>

int main(void)
{
  int a;
  a = -1;

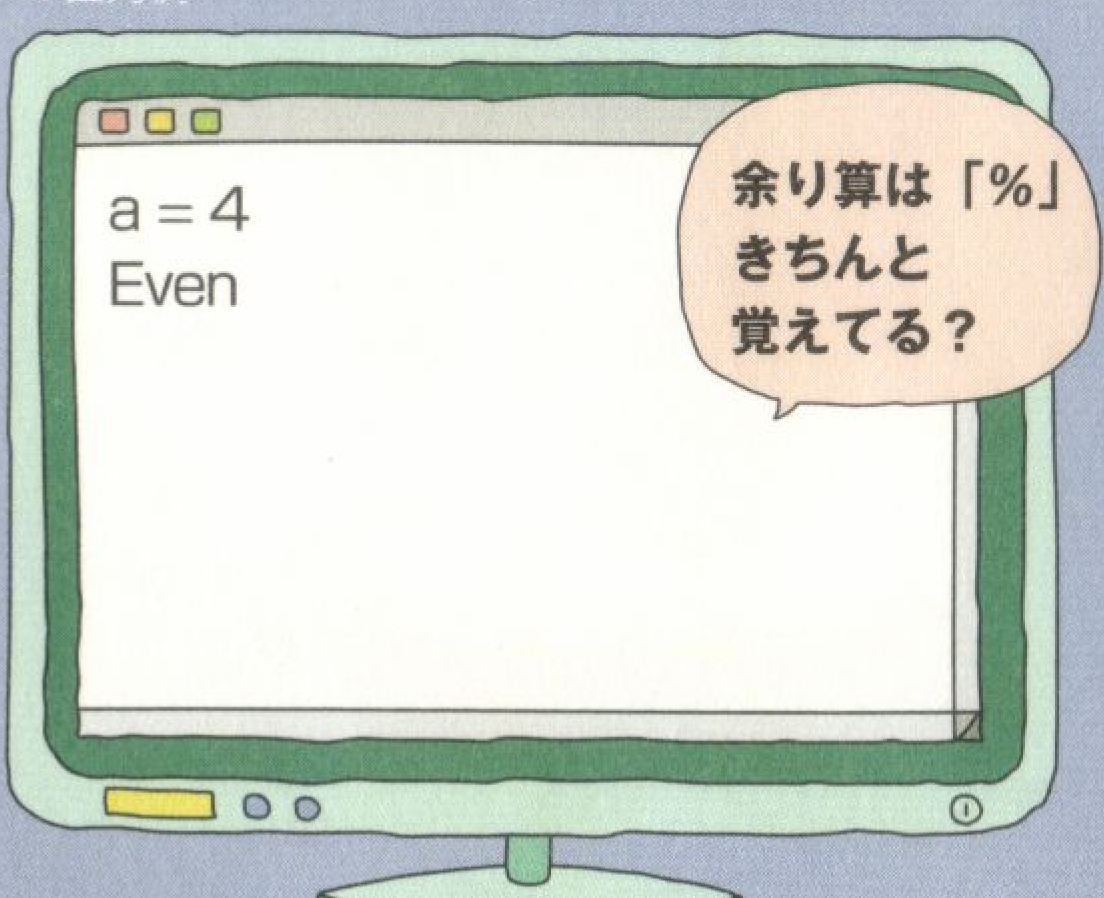
  printf("a = %d¥n", a);

  if (a >= 0){
        printf("%d >= 0¥n", a);
  }else{
        printf("%d < 0¥n", a);
  }

  return 0;
}
```

例題　条件分岐❷if-else文

▼出力例



int型の変数aとint型の変数bを用意します。aに値を代入して、bにaの値を2で割った余りを代入して、aが偶数ならば「Even」、そうでなければ「Odd」というメッセージを出力するソースコードを書いてください。

●解答

```
/* Exercise8.c */
#include <stdio.h>

int main(void)
{
  int a;
  int b;

  a = 4;
  b = a % 2;

  printf("a = %d¥n", a);

  if (b == 0){
         printf("Even¥n");
  }else{
         printf("Odd¥n");
  }

  return 0;
}
```



▶専門的な説明

例題のソースコードは、int型の変数aだけで、次のように書くことができます。

```
 1:/* Exercise81.c */
 2:#include <stdio.h>
 3:
 4:int main(void)
 5:{
 6:  int a;
 7:  a = 4;
 8:
 9:  printf("a = %d¥n", a);
10:
11:  if ((a%2) == 0){ ──→ 条件には式を使うこともできます
12:        printf("Even¥n");
13:  }else{
14:        printf("Odd¥n");
15:  }
16:
17:  return 0;
18:}
```

11行目を見てください。「a%2」のように、条件には、その比較対象に「式」を使うこともできます。

例題 条件分岐❸条件の組み合わせ

次の条件を論理演算子で書いてください。

❶「変数aは、0以上かつ10以下」
❷「変数aは、0以上かつ10未満」
❸「変数aは、0より大きいかつ10以下」
❹「変数aは、0ではない」
❺「変数aは、0より大きいまたは10未満」



●解答

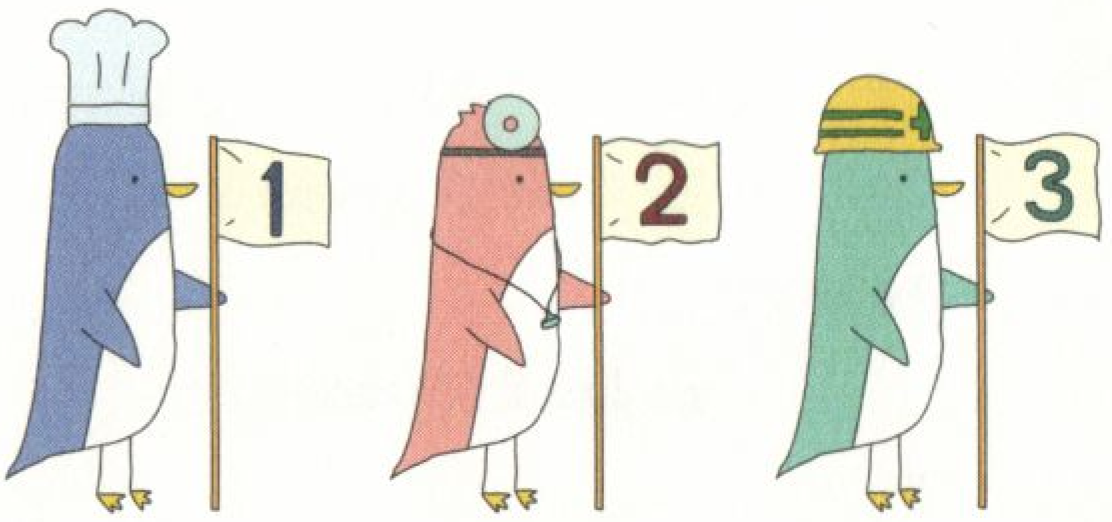
```
❶ a>=0 && a<=10
❷ a>=0 && a<10
❸ a>0 && a<=10
❹ a != 0
❺ a>0 || a<10
```

▶▶switch文とbreak文

変数の値によって処理を選択したい場合には**switch文**を使います。

if-else文でも同じことができるのですが、こちらのほうが書きやすい場合もあります。

例えば、3択問題で、キーボードから「1,2,3」の1文字を入力として受け取って処理を分岐させたい場合、などがこれにあたります。



書式

```
switch(変数){
     case 値1:
          文1;  ──→ 値1のときに実行する文
           ⋮
          break;
     case 値2:
          文2:  ──→ 値2のときに実行する文
           ⋮
          break;
     default:
          文3:  ──→ 値1でも値2でもないときに実行する文
           ⋮
}
```

●解説

1行目のカッコの中にはキーになる変数を書きます。変数の値が値1のとき、3行目から4行目を実行し、5行目の**break文**で14行目に進みます。変数の値が値1でなく、値2のときに、7行目から8行目を実行し、9行目で14行目に進みます。変数の値がどの値でもないときは、10行目の**default:**より下の11行目と12行目を実行し、14行目に進みます。

それでは、実際にC言語で「switch文」を使ってみましょう。

```
/* MyC6.c */
#include <stdio.h>

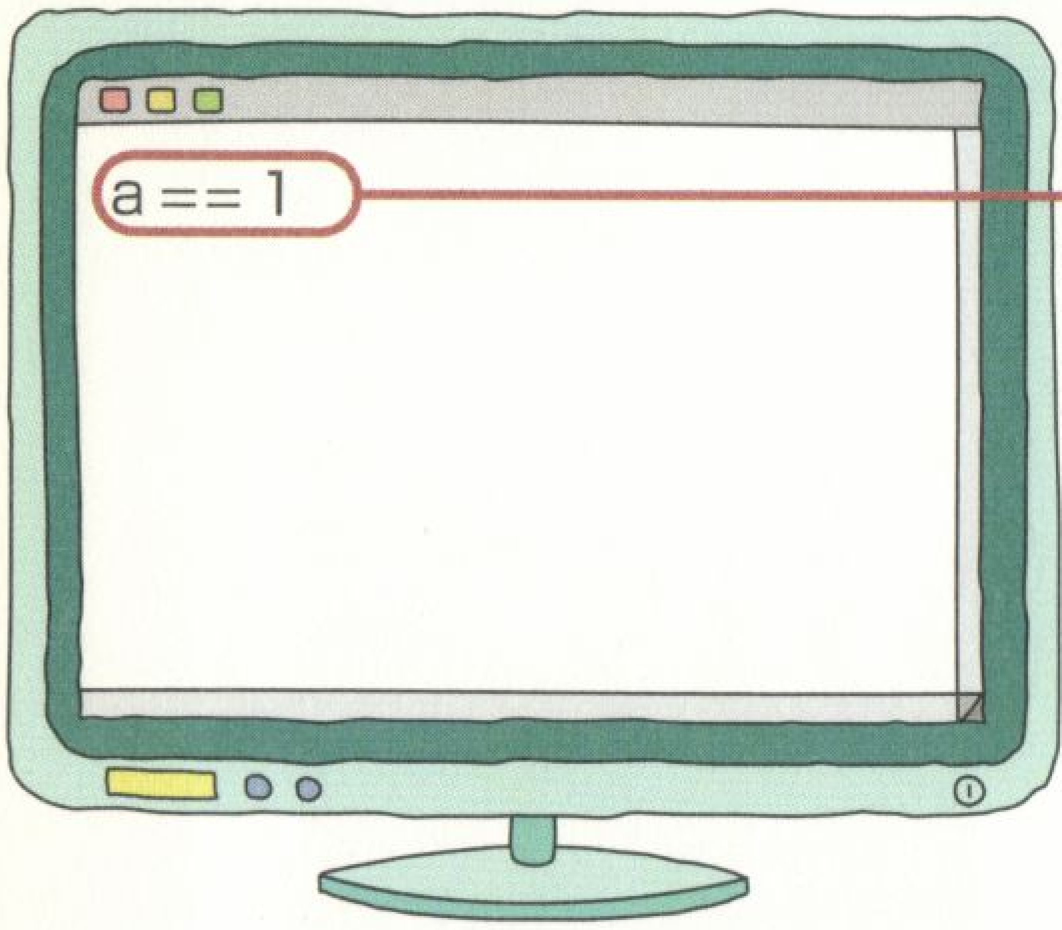
int main(void)
{
  int a;
  a = 1;  ──→ aの値は1です

  switch (a) {
        case 0:
                      printf("a == 0¥n");
                      break;
        case 1:  ──→ aの値は1ですので、ここでキャッチされます
                      printf("a == 1¥n");
                      break;
        default:
                      printf("a != 0 and a != 1¥n");
  }

  return 0;
}
```

▶作成手順と実行結果

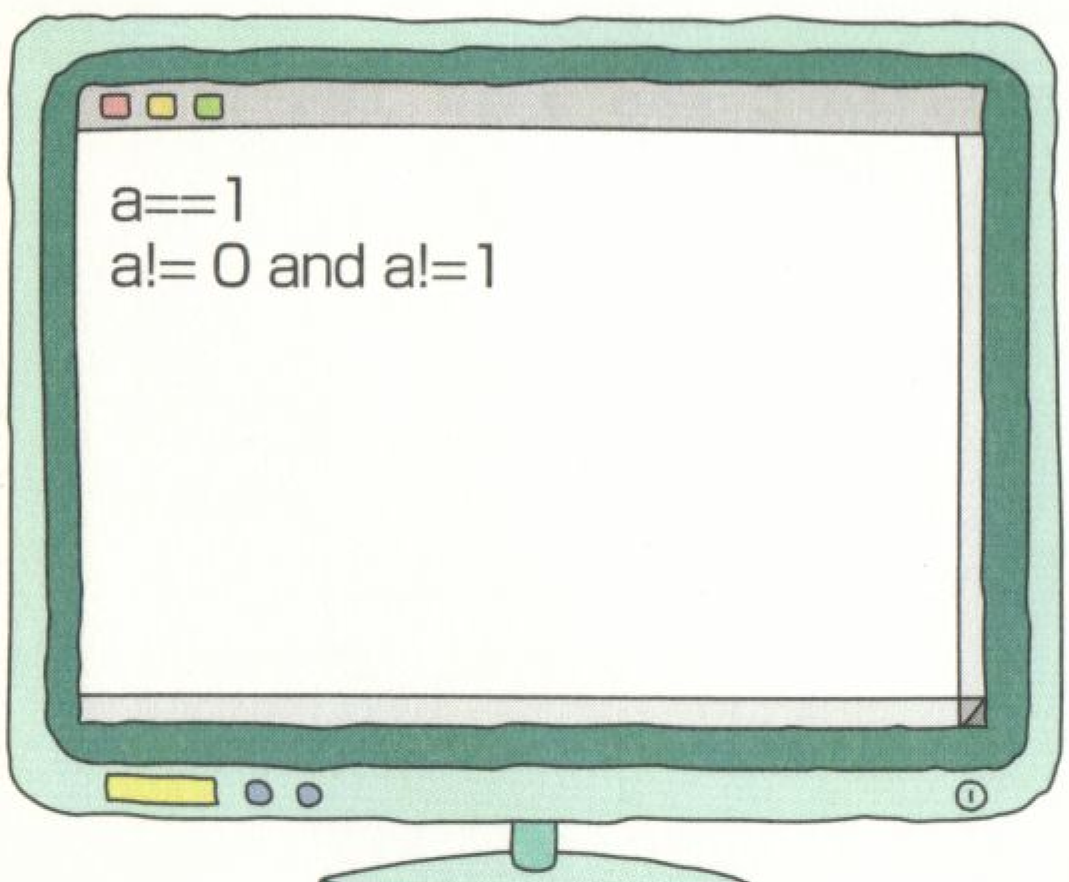
❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC6.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC6.c -o MyC6** Enter」と入力します。
❻「**MyC6** Enter」と入力します。



7行目で、a=1となっているので、
13行目でキャッチされる
よって、画面には「a ==1」と表示される

例題 条件分岐❹break文

　サンプルソースコード「MyC6.c」の、15行目のbreak文を消したときの結果を考えてください。



●解答

　break文を忘れると、変数の値が比較する値であろうとなかろうと、下の文を実行していってしまいますので、注意しましょう。



例題 条件分岐❺switch文

int型の変数aに値を代入して、aの値によって、値が0のときは「×」、1のときは「○」、それ以外のときは「?」、というメッセージが画面に出力するソースコードを書いてください。

●解答

```
//Exercise10.c
#include <stdio.h>

int main(void)
{
  int a;
  a = 1;

  switch (a) {
        case 0:
              printf("x¥n");
              break;
        case 1:
              printf("o¥n");
              break;
        default:
              printf("?¥n");
  }
}
```

Section 02 プログラムの案内人②

C言語で用意されている制御文には、「条件分岐」の他に「繰り返し」があります。「繰り返し」とは、同じ動作を何度も続けることです。また、「繰り返し」といっしょに使うことの多い「配列」というデータの扱い方があります。ここでは、それを説明していきます。

繰り返し

条件が成立する間、あるブロック内を繰り返し実行するのが**繰り返し**です。繰り返しには、**for文**、**while文**、**do-while文**、**continue文**があります。

先ほどの例で考えると、「コピーを取る」のように何回も同じことを繰り返すときに使えばよいでしょう。

1:「3人ぶん（Aさん、Bさん、Cさん）のコピーを取る」
2: コピー室へ行く
3: 3人ぶんのコピーを取る
4: 席へ戻る



それでは、実際にC言語で、繰り返しを使ってみましょう。

▶▶for文

for文は、回数をカウントしながら、繰り返して同じことをするための文です。そのため、繰り返した回数（カウンタ）を数えるための変数を使います。

```
for (カウンタに初期値を代入; 繰り返すための条件; 回数のカウントなど) {
        文1; → 繰り返し実行する文
        文2; → 繰り返し実行する文
         ⋮
}
```

書式

for文の流れは、実際にC言語のソースコードを見たあとのほうがわかりやすいと思いますので、先にソースコードを見てみましょう。

```
/* MyC7.c */
#include <stdio.h>

int main(void)
{
  int i;

  for (i=1; i<=3; i++) {
          printf("%d¥n", i);
  }

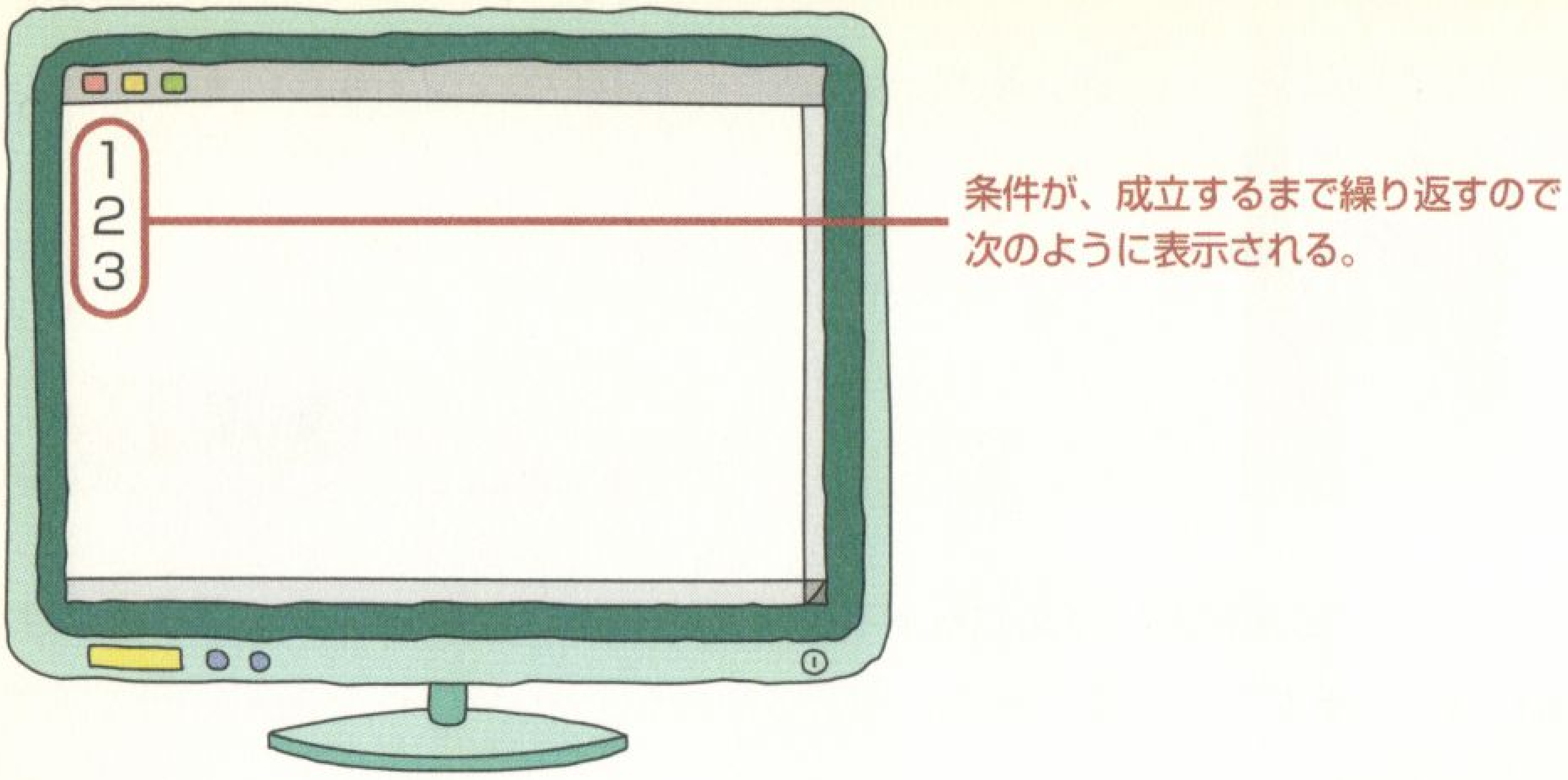
  return 0;
}
```

カウンタには「i」がよく使われます

「i++」は、算術演算子の一つで i=i+1と同じことです

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC7.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC7.c -o MyC7** Enter」と入力します。
❻「**MyC7** Enter」と入力します。



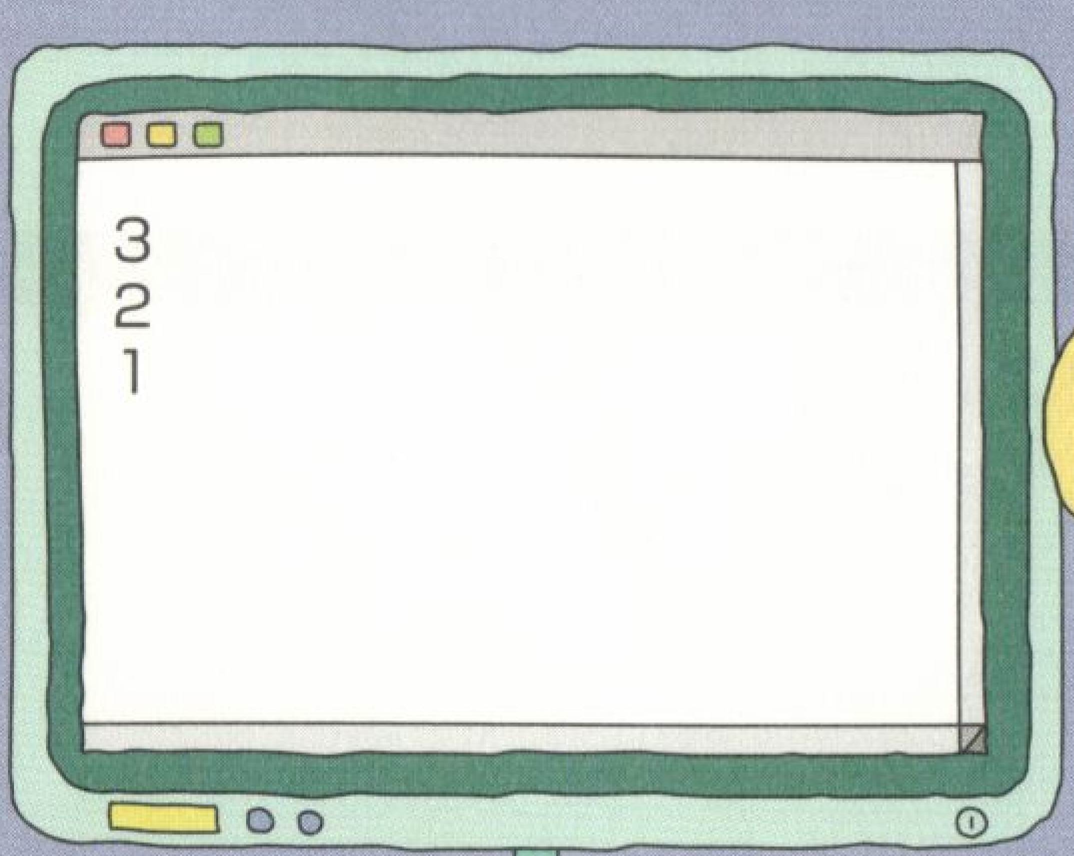
8行目から10行目で繰り返しをしていますが、このときのfor文の流れを実行結果を元に表にすると、次のようになります。

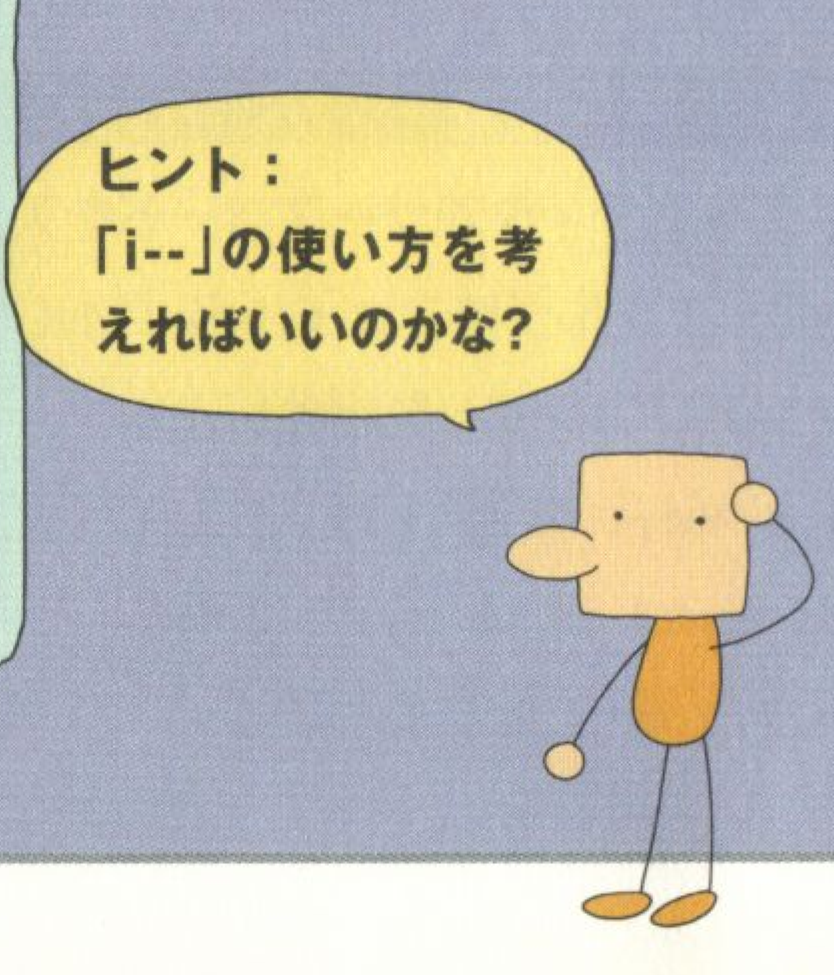
▼for文の流れ

変数iの値	条件 i <= 3	i++	内容
1	成立する	i = 2	繰り返す➡1回目繰り返す
2	成立する	i = 3	繰り返す➡2回目繰り返す
3	成立する	i = 4	繰り返す➡3回目繰り返す
4	成立しない	i = 3	終わる

例題 繰り返し❶for文

次のように、画面に出力するソースコードを書いてください。





●解答

```
/* Exercise11.c */
#include <stdio.h>

int main(void)
{
  int i;

  for (i=3; i>=1; i--) {
        printf("%d¥n", i);
  }

  return 0;
}
```

→「i>=1」を「i>0」にしても同じことです

繰り返しの中では、さらに繰り返しをすることができます。これを使うと、簡単に九九の計算（説明を簡単にするため2の段までです）もできるのです。

これも、実際にC言語のソースコードを見たあとのほうがわかりやすいと思いますので、先にソースコードを見てみましょう。

```
/* MyC8.c */
#include <stdio.h>

int main(void)
{
  int i;
  int j;

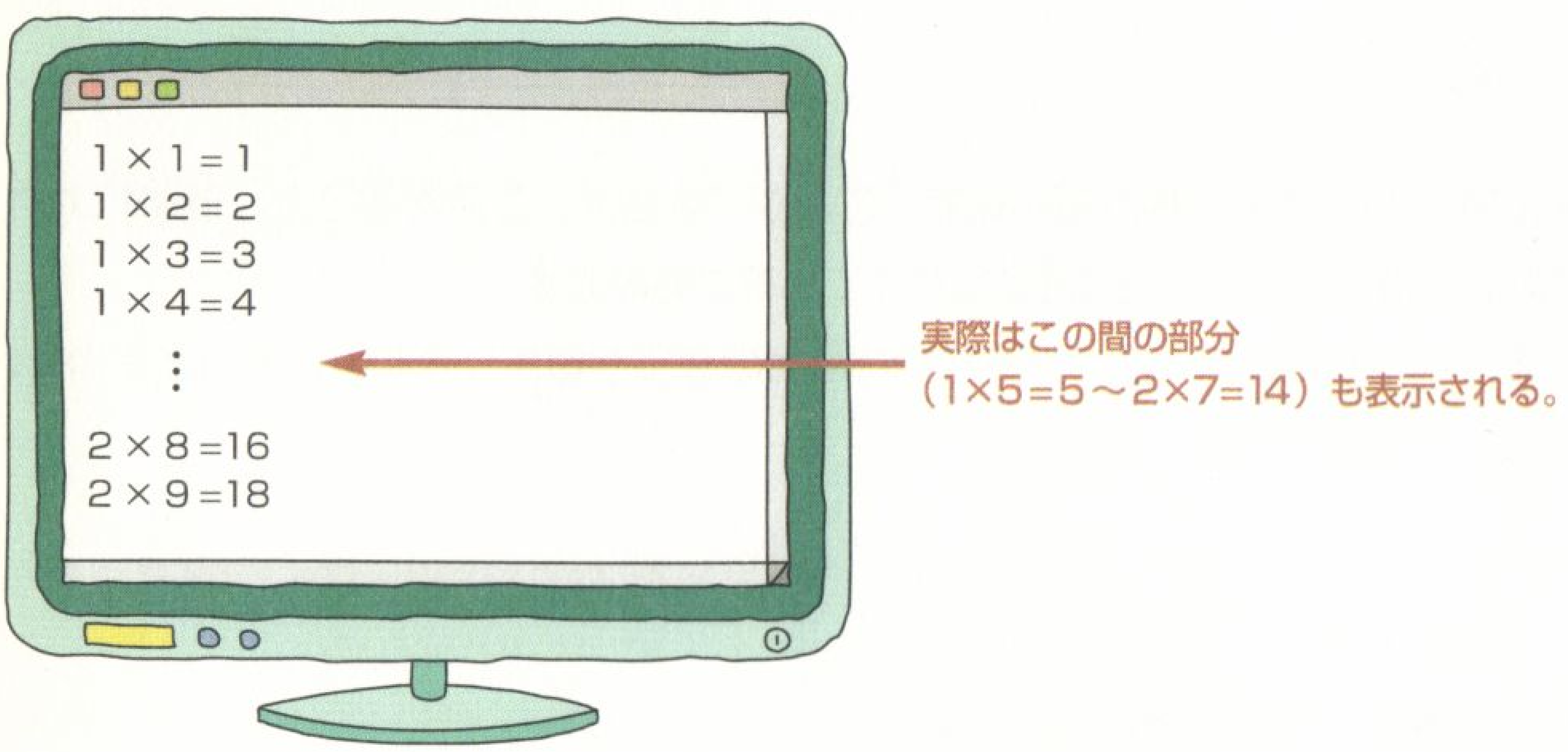
  for (i=1; i<=2; i++) {
        for (j=1; j<=9; j++){
              printf("%d x %d = %d¥n", i, j, (i*j));
```

「i*j」という式の値が画面に出力されます

```
12:             }
13:     }
14:
15:     return 0;
16:}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC8.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「`cd c:¥MyC` Enter」と入力してください。
❺「`gcc MyC8.c -o MyC8` Enter」と入力します。
❻「`MyC8` Enter」と入力します。



●解説

9行目から12行目で、2つのfor文の繰り返しをしています。流れを実行結果を元に説明しましょう。

▼外側のfor文

変数iの値	条件 i <= 3	i++	外側のfor文
1	成立する	i = 2	繰り返す➡内側のfor文1へ
2	成立する	i = 3	繰り返す➡内側のfor文2へ
3	成立しない	i = 4	終わる

▼内側のfor文❶

変数jの値	条件 j <= 9	j++	内側のfor文（i = 1）
1	成立する	j = 2	繰り返す➡1かける1は1
2	成立する	j = 3	繰り返す➡1かける2は2
3	成立する	j = 4	繰り返す➡1かける3は3
4	成立する	j = 5	繰り返す➡1かける4は4
5	成立する	j = 6	繰り返す➡1かける5は5
6	成立する	j = 7	繰り返す➡1かける6は6
7	成立する	j = 8	繰り返す➡1かける7は7
8	成立する	j = 9	繰り返す➡1かける8は8
9	成立する	j = 10	繰り返す➡1かける9は9
10	成立しない	j = 3	終わる　➡外側のfor文へ

▼内側のfor文❷

変数jの値	条件 j <= 9	j++	内側のfor文（i = 2）
1	成立する	j = 2	繰り返す➡2かける1は2
2	成立する	j = 3	繰り返す➡2かける2は4
3	成立する	j = 4	繰り返す➡2かける3は6
4	成立する	j = 5	繰り返す➡2かける4は8
5	成立する	j = 6	繰り返す➡2かける5は10
6	成立する	j = 7	繰り返す➡2かける6は12
7	成立する	j = 8	繰り返す➡2かける7は14
8	成立する	j = 9	繰り返す➡2かける8は16
9	成立する	j = 10	繰り返す➡2かける9は18
10	成立しない	j = 3	終わる　➡外側のfor文へ

▶▶while文

while文でもfor文と同じことができますが、繰り返す条件が繰り返し回数とは限りません。条件が成立する間、ブロックの中を繰り返します。これは、何回繰り返したらよいのか、よくわからない場合にも便利です。

先ほどの例で考えると、コピーの必要な人が、A、B、Cさんだけではなくてもっとたくさんいるのだけれど、具体的な人数がわからない（コピー用紙がある限りコピーを取らなければならない）ときなどに使えばよいでしょう。

1:「呼ばれるまでコピーを取る」
2: コピー室へ行く
3: コピーを取る
4: 呼ばれたら、席へ戻る



書式

```
while (条件) {
      文1;   ──→ 繰り返し実行する文
      文2;   ──→ 繰り返し実行する文
       ⋮
}
```

それでは、実際にC言語でwhile文を使ってみましょう。

```
/* MyC9.c */
#include <stdio.h>

int main(void)
{
  int i;
```

```
    i = 1;

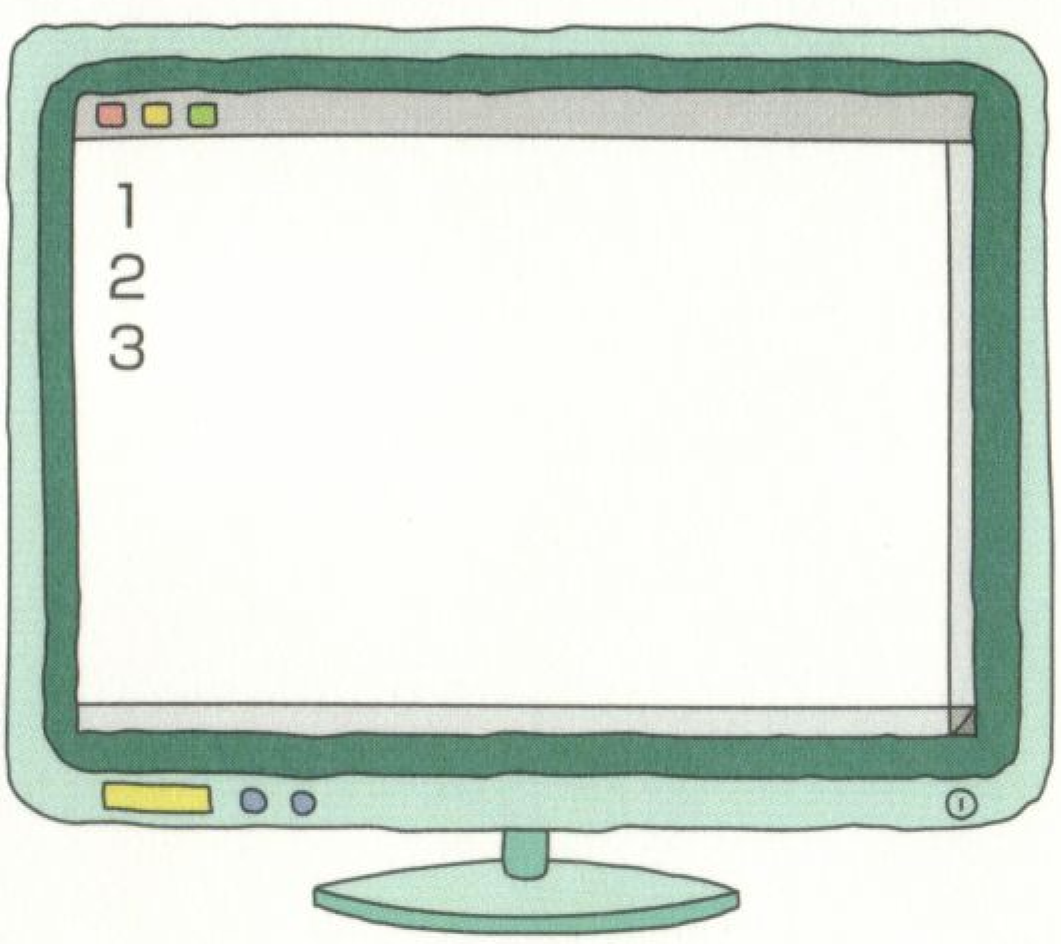
    while (i<=3) {
            printf("%d¥n", i);
            i++;
    }

    return 0;
}
```

→ iが3以下の間、ブロック内を実行します

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC9.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC9.c -o MyC9** Enter」と入力します。
❻「**MyC9** Enter」と入力します。





●解説

実行結果を見ればわかるように、このサンプルソースコード「MyC9.c」は、for文のサンプルコード「MyC7.c」と同じ結果を出力するものです。

▶▶do-while文

do-while文は、while文とほとんど同じです。違うのは、while文が繰り返しの条件を最初に書くのに対して、do-while文は最後に書く点です。

つまり、do-while文では、条件が成立しなくても1回はブロック内を実行するということです。

書式

```
do {
        文1; → 繰り返し実行する文
        文2; → 繰り返し実行する文
         :
} while(条件); ——→「;」を忘れないようにしましょう
```

それでは、実際にC言語で「do-while文」を使ってみましょう。

```
/* MyC10.c */
#include <stdio.h>

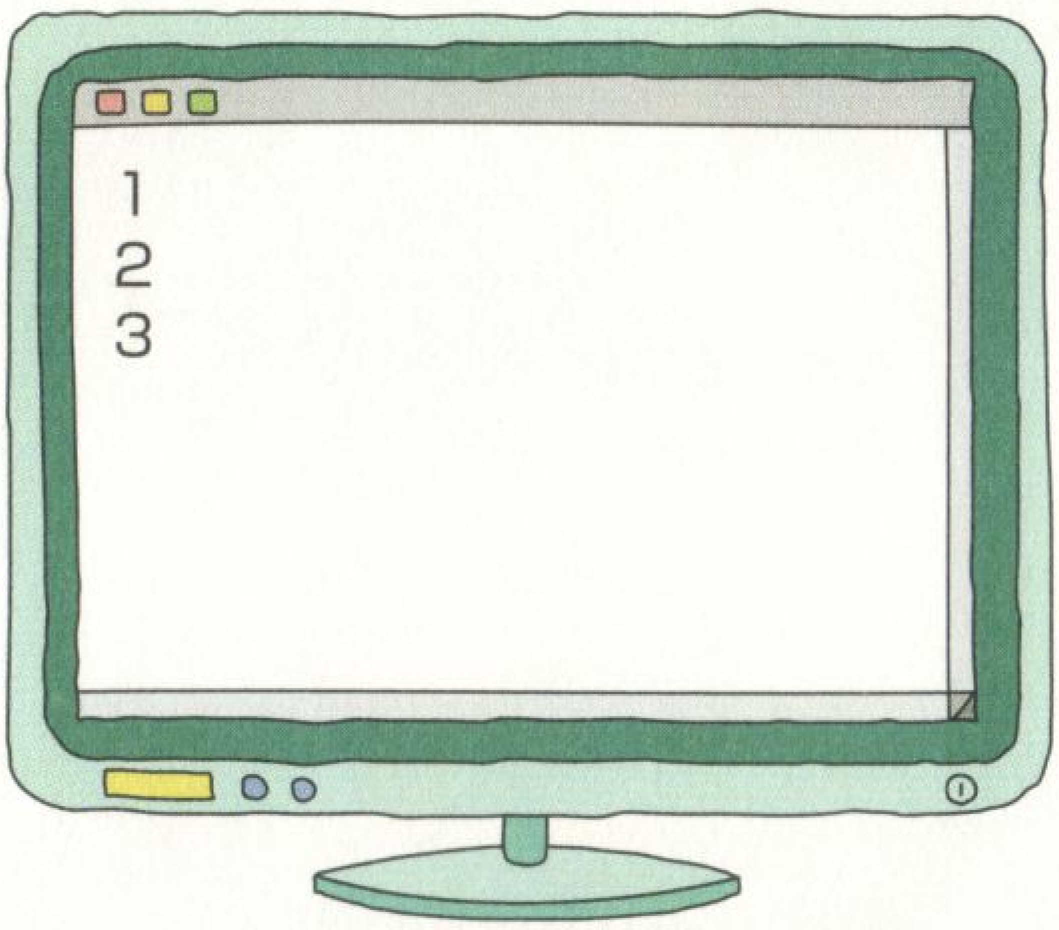
int main(void)
{
  int i;
  i = 1;

  do {
        printf("%d¥n", i);
        i++;
  } while(i<=3);

  return 0;
}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC10.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥Myc** Enter」と入力してください。
❺「**gcc MyC10.c -o MyC10** Enter」と入力します。
❻「**MyC10** Enter」と入力します。





●解説

実行結果をみればわかるように、このサンプルソースコード「MyC10.c」もfor文のサンプルコード「MyC7.c」と同じ結果を出力するものです。

例題　繰り返し❷while文とdo-while文

サンプルソースコード「MyC9.c」と「MyC10.c」の7行目を「i = 4」として、その実行結果の違いを確認してください。

●解答

▶「MyC9.c」の場合

何も表示されません。

▶「MyC10.c」の場合

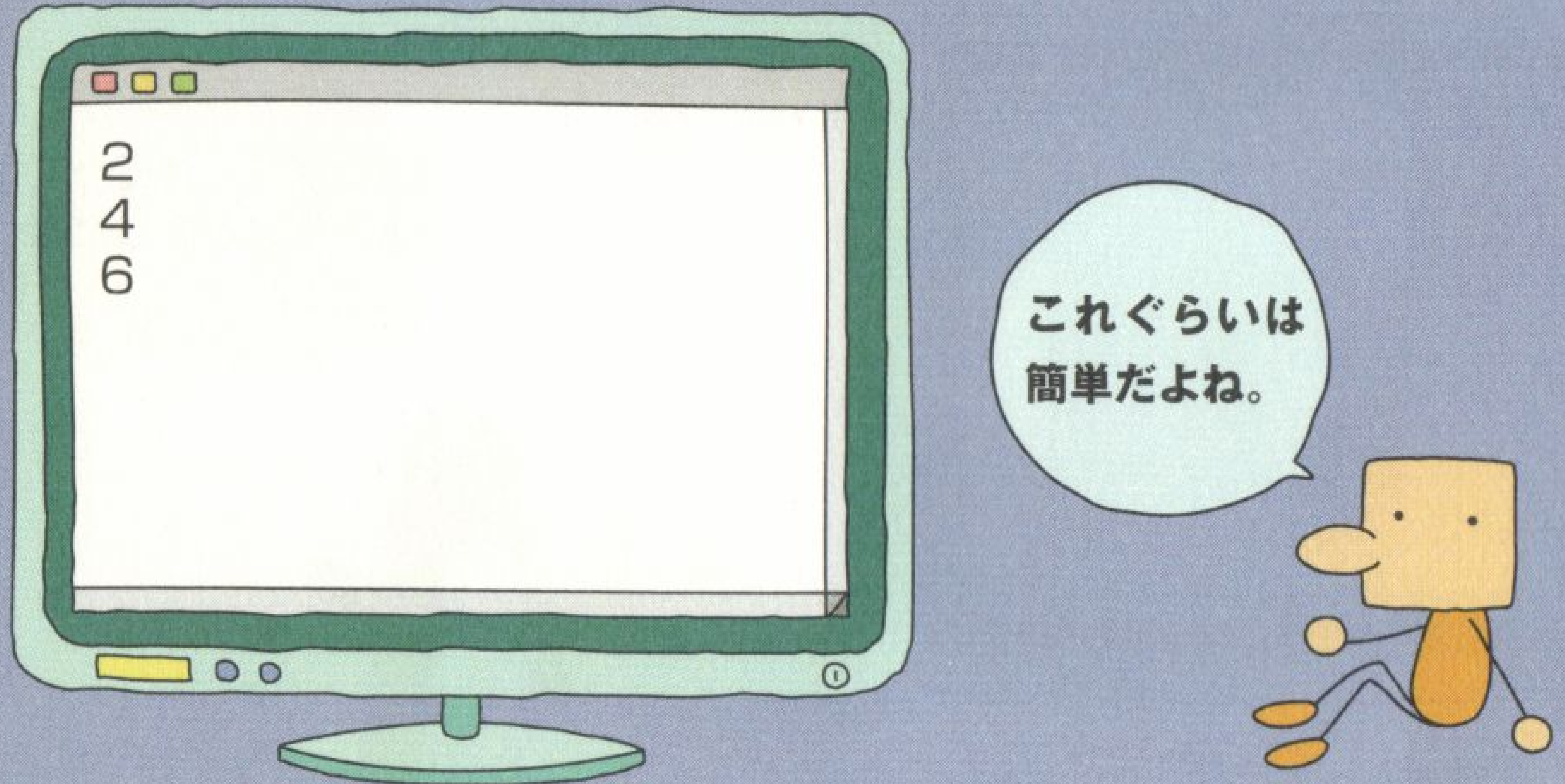
「4」と出力されます。

●解説

先ほど説明したように、do-while文では、条件が成立しなくても1回はブロック内を実行するということが実感できたと思います。

例題　繰り返し❸実力試し

次のように画面に出力するコードを書いてください。



●解答

この問題は、解答の仕方が様々あります。そのため、ここでは解答を用意しません。
いままでやってきたことを通して、自分でソースコードを書いてみてください。

配列❶

1つの変数には、1つのデータしか入れることはできません。しかしこれでは、全校生徒の身長や体重の平均を知りたいというときのように、同じ性質や意味を持つ大量のデータを処理するときには困ってしまいます。

全校生徒ぶんの変数宣言も面倒ですし、変数名も全校生徒ぶんを書かなくてはなりません。



```
int a1;
int a2;
      :
int a100;

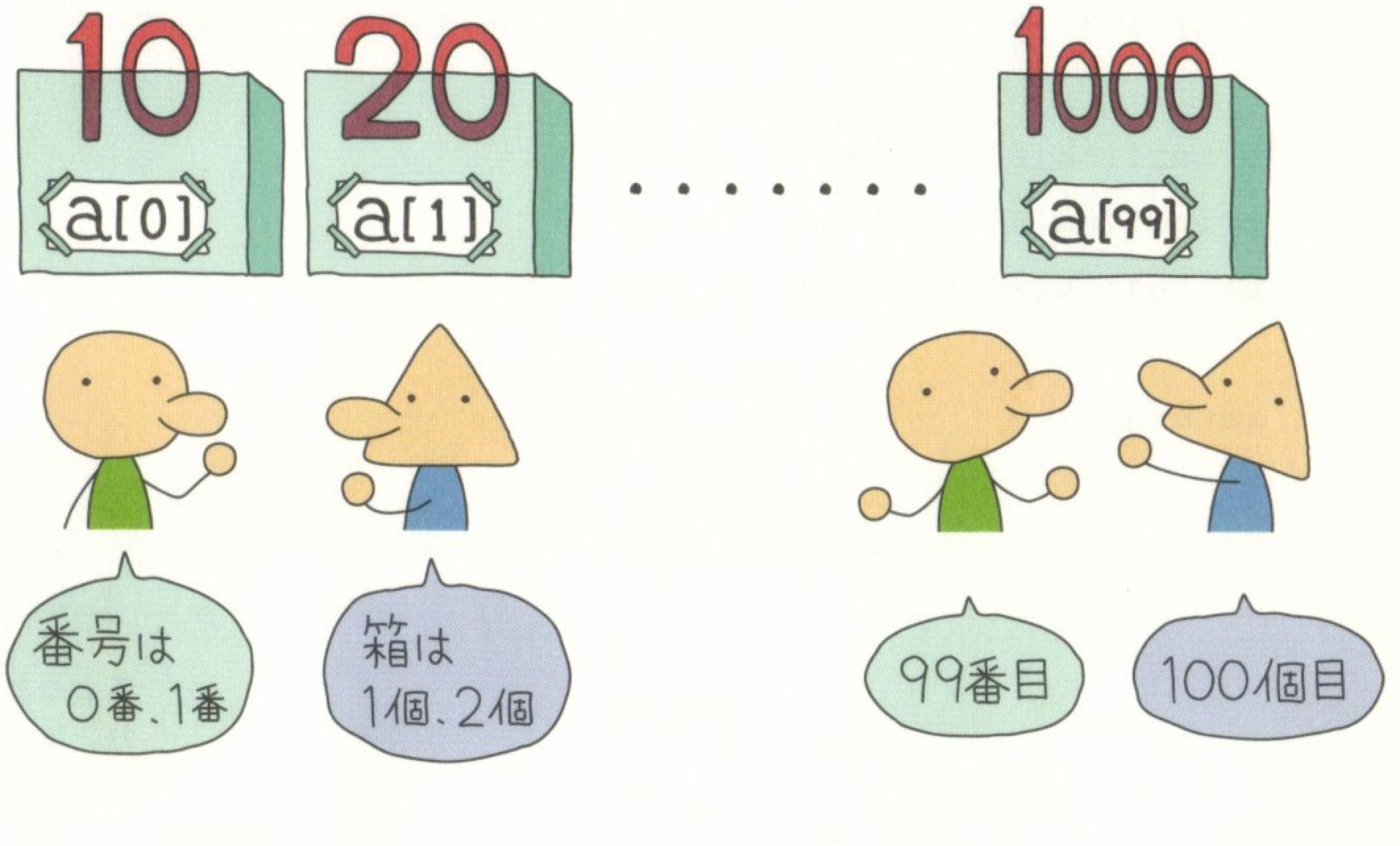
a1 = 10;
a2 = 20;
      :
a100 = 1000;
```

100個の変数をすべて宣言する場合

配列は、このようなときに便利です。

配列は、同じ型の変数（箱）が組みになっているようなものです。組になった変数全体に付ける名前を**配列名**といい、個々の変数（箱）は**添え字**（そえじ）という番号で区別します。添え字は「0,1,2･･･」という具合に付いていきます。

また、配列を使うには、組になる変数（箱）をいくつ用意すればよいか、あらかじめ決めておく必要があります。その数のことを、**要素数**といいます。



配列も変数と同じで、「宣言」してから使います。

型　配列名[要素数]　**書式**

▶例

```
int a [100]
```

型　配列名　要素数

それでは、実際にC言語で「配列」を使ってみましょう。

```
/* MyC11.c */
#include <stdio.h>

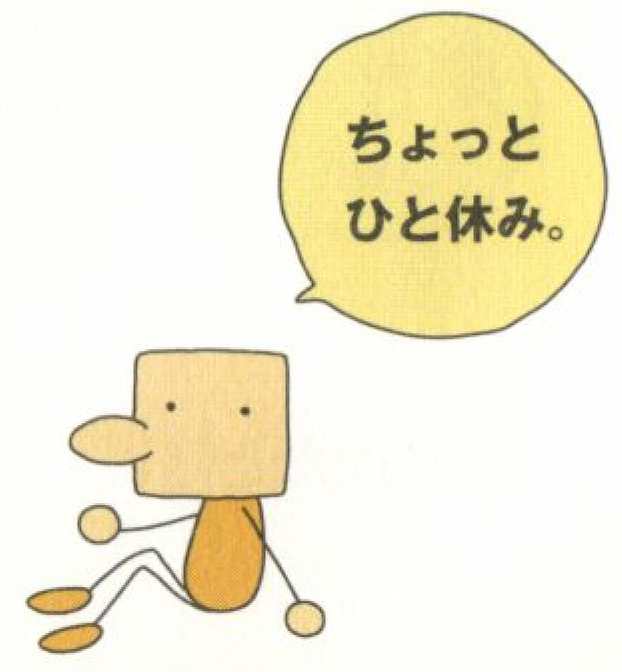
int main(void)
{
  int a[3];

  a[0] = 1;
  a[1] = 2;
  a[2] = 3;

  printf("a[0] = %d¥n", a[0]);
  printf("a[1] = %d¥n", a[1]);
  printf("a[2] = %d¥n", a[2]);

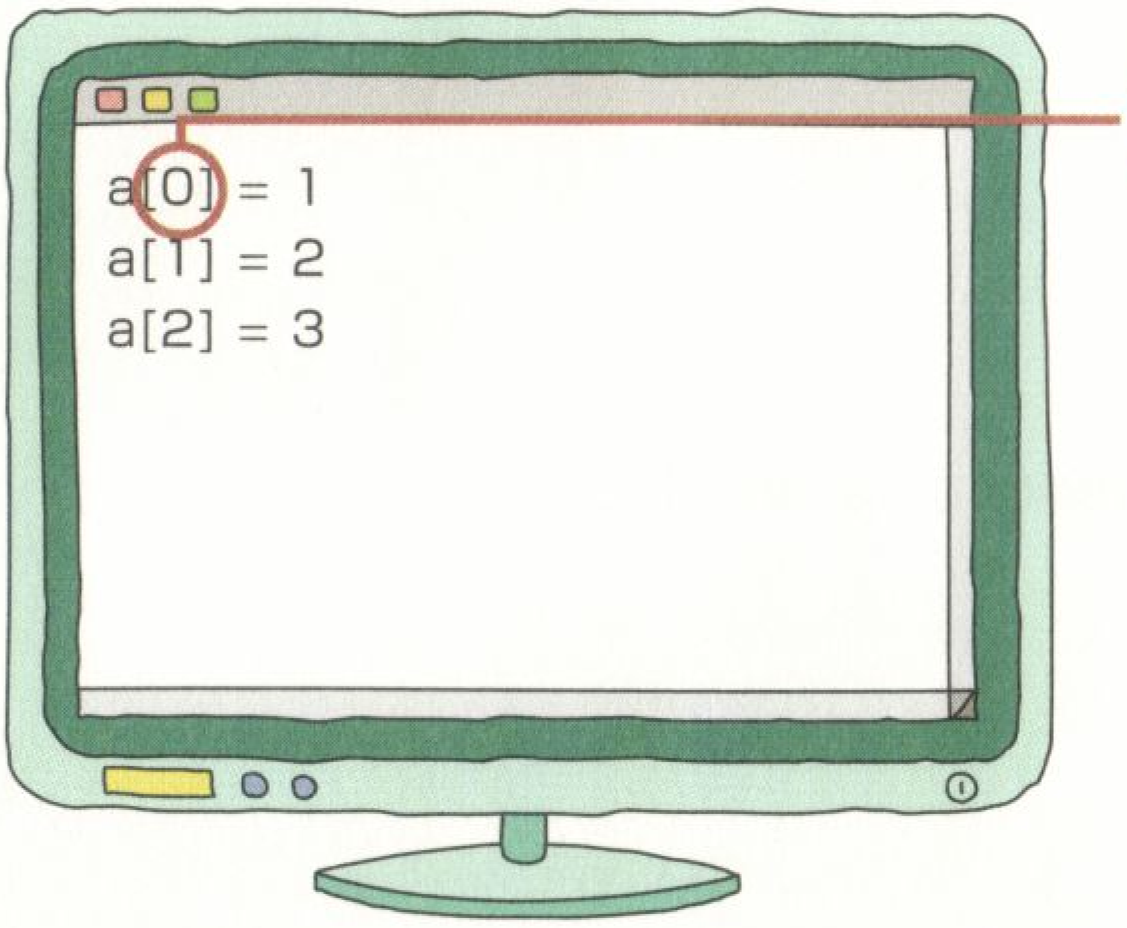
  return 0;
}
```

- int a[3]; → 配列の宣言
- a[0] の 0 → 添え字
- a[0] = 1; → 配列の要素に値を代入、添え字が「0」であることに注意してください
- a[1] = 2; → 配列の要素に値を代入
- a[2] = 3; → 配列の要素に値を代入、添え字が「2」であることに注意してください



▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC11.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC11.c -o MyC11** Enter」と入力します。
❻「**MyC11** Enter」と入力します。



●解説

注意しなければならないのは、6行目の要素数が「3」であるのに、10行目の添え字が「2」であることです。要素数が「3」でも、8行目の添え字を見ると「0」から始まっているので、10行目の添え字は「2」になるのです。

このことは、組になる変数（箱）を3個用意しても、添え字は、必ず「2」で終わることになる（「3」以上を添え字としては使えない）ということです。

例題 配列と繰り返し

サンプルソースコードの「MyC11.c」と、同じ結果を画面に出力するソースコードを配列と繰り返しを使って書いてください。

●解答

```
 1:/* Exercise13.c */
 2:#include <stdio.h>
 3:
 4:int main(void)
 5:{
 6:  int a[3];
 7:  int i;
 8:
 9:  for (i = 0; i<=2; i++) {
10:          a[i] = i + 1;
11:          printf("a[%d] = %d¥n", i, a[i]);
12:  }
13:
14:  return 0;
15:}
```

●解説

解答と、必ずしも同じでなくてもかまいません。ポイントは、9行目の配列の添え字に、for文のカウンタ「i」を使ったらどうだろうと、考えたかどうかです。

また、このように配列と繰り返しを組み合わせると、カウンタの値が100でも1000でも、8行目から11行目は、for文の条件である「i<=2」の部分が「i<=100」、「i<=1000」と変わるだけで、行数は変わりません。

今回は、配列の要素数が3個と少ないので実感しにくかったかも知れませんが、もしこれが、100個や1000個だったら、ソースコードを書く手間が、繰り返しを使うことで格段に楽になることがわかると思います。

配列❷

ここでは、配列の使い方について、もう少し具体的に考えてみましょう。

例えば、次のようなことを考えてみてください。

3年A組の、生徒5人（A0さん、A2さん、・・・、A5さん）の体重を表示するプログラムを作りたいとします。ソースコードは次のようになります。

```
/* MyC111.c */
#include <stdio.h>

int main(void)
{
  int i;
  int A[5];

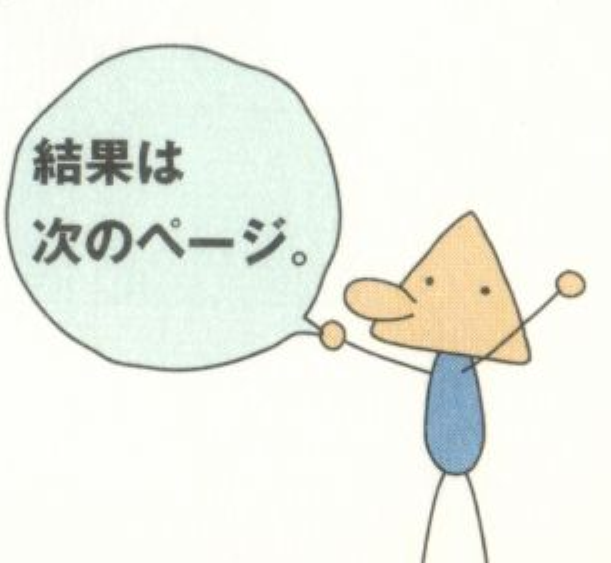
  A[0] = 20;
  A[1] = 21;
  A[2] = 22;
  A[3] = 23;
  A[4] = 24;

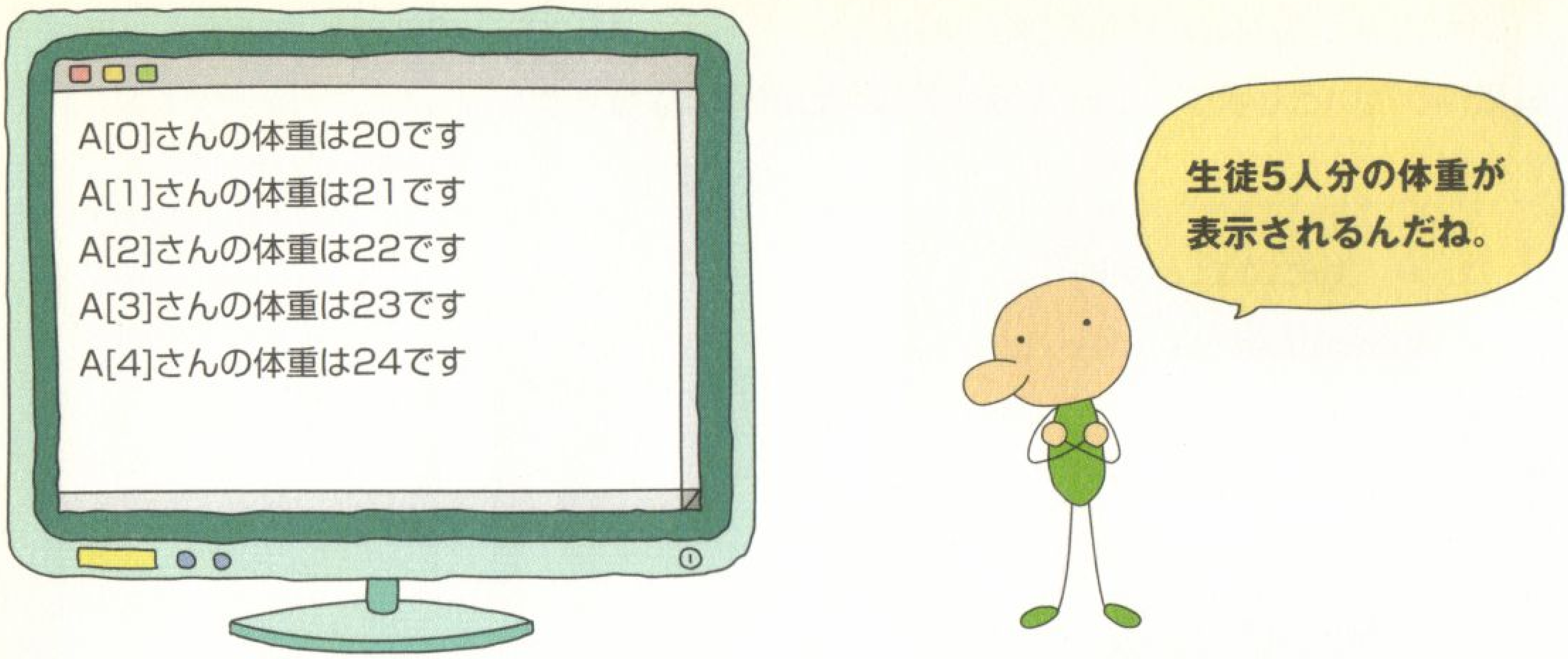
  for (i = 0; i<=4; i++) {
              printf("A[%d]さんの体重は%dです¥n", i, A[i]);
  }

  return 0;
}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC111.C」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC111.c -o MyC111** Enter」と入力します。
❻「**MyC111** Enter」と入力します。

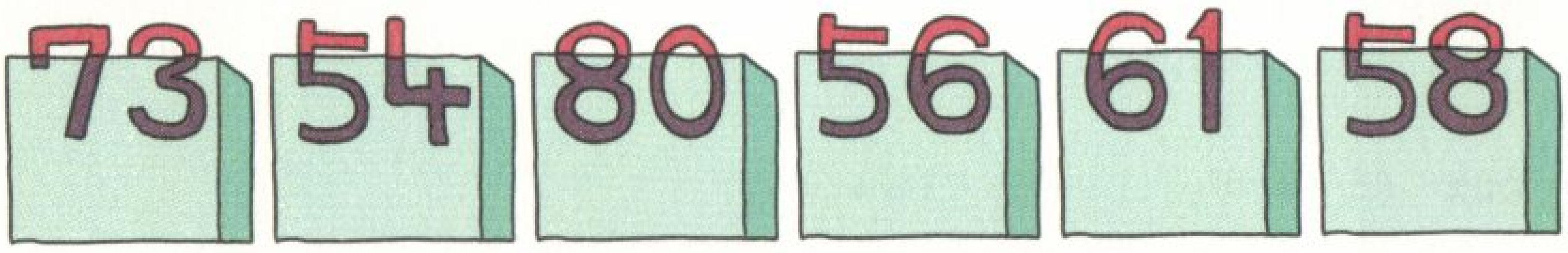




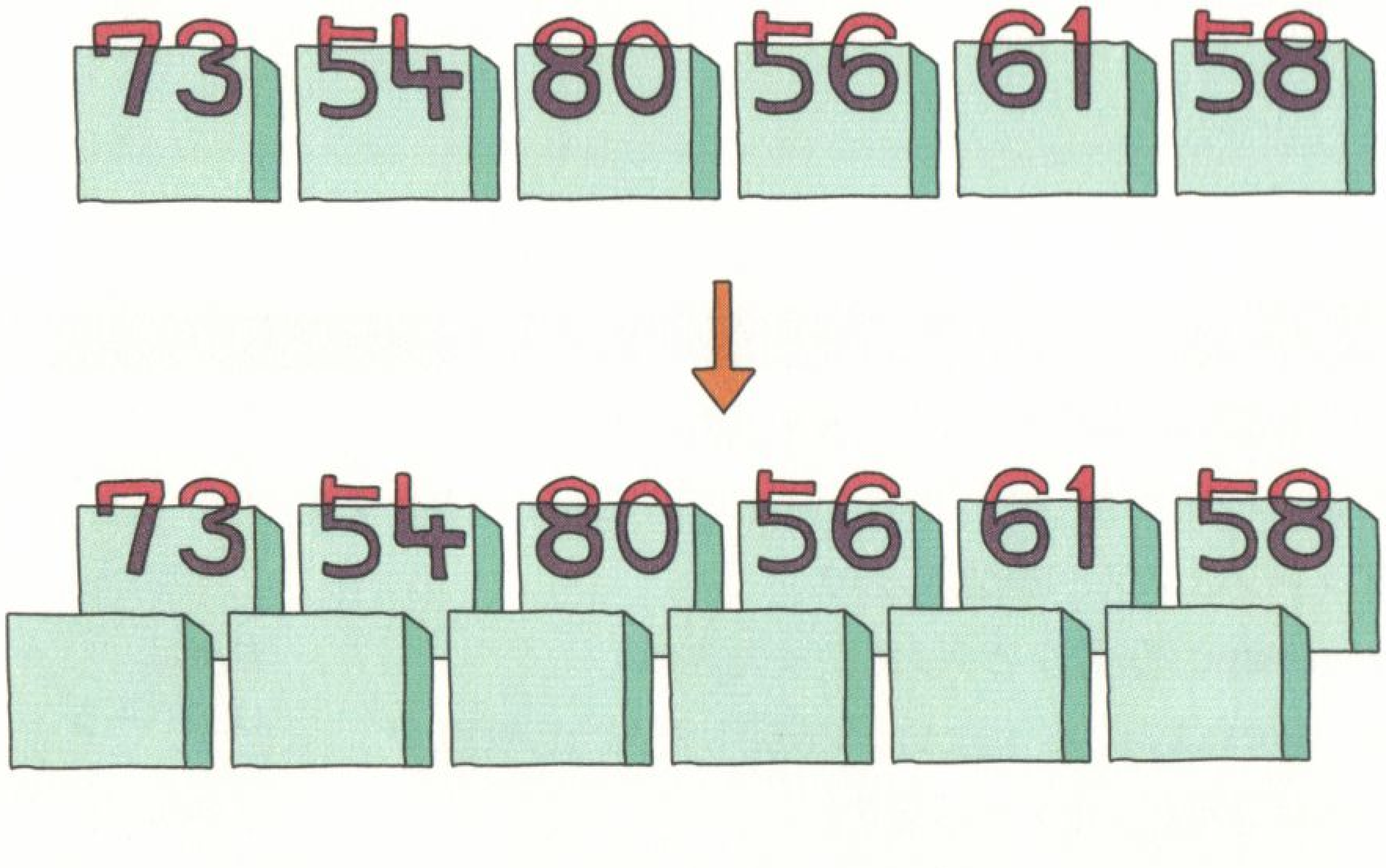
さて、ここで生徒の体重だけではなく、身長も表示したいと考えたとき、どうしたらよいでしょうか？　a[X]には、体重だけしか代入できそうにありません。

そういうときのために、配列をもっと便利に使う方法があるのです。

配列は、同じ型の変数（箱）が組みになっているものだと説明したのですが、これまで説明したものは、箱を横に1列に並べたものでした。



この列を、2列にしたらどうなるでしょう。



そして、この2列目を、身長の表示に使ってやればよいのです。このような配列を**2次元配列**といいます。ちなみに、いままでの配列は**1次元配列**といいます。

それでは、「2次元配列」を使って、ソースコードを書き直してみましょう。

```
/* MyC112.c */
#include <stdio.h>

int main(void)
{
  int i;
  int A[5][2];

  A[0][0] = 20;
  A[0][1] = 120;
  A[1][0] = 21;
  A[1][1] = 121;
  A[2][0] = 22;
  A[2][1] = 122;
  A[3][0] = 23;
  A[3][1] = 123;
  A[4][0] = 24;
  A[4][1] = 124;

  for (i = 0; i<=4; i++) {
        printf("A[%d]さん¥n", i);
        printf("体重:%dで身長:%d¥n", A[i][0], A[i][1]);
  }

  return 0;
}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。

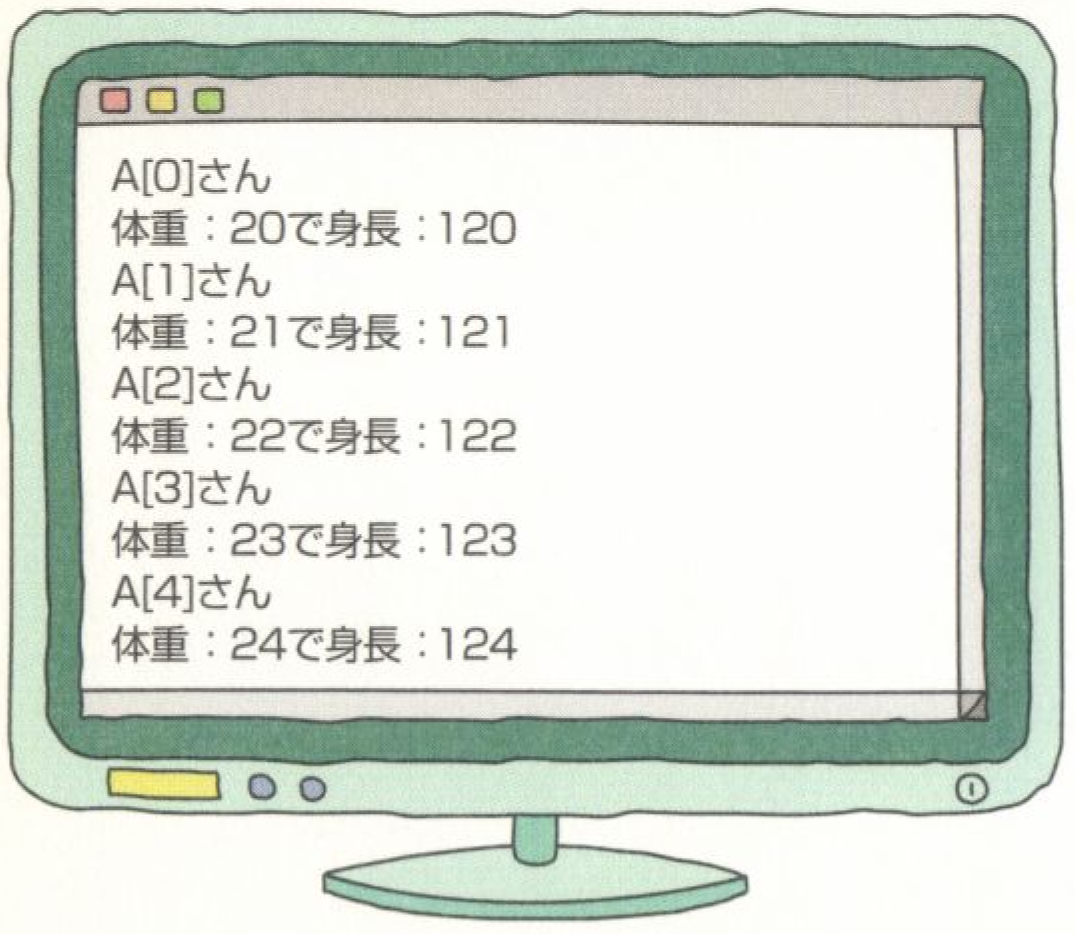
❷「MyC」フォルダにファイル名「MyC112.C」で保存します。

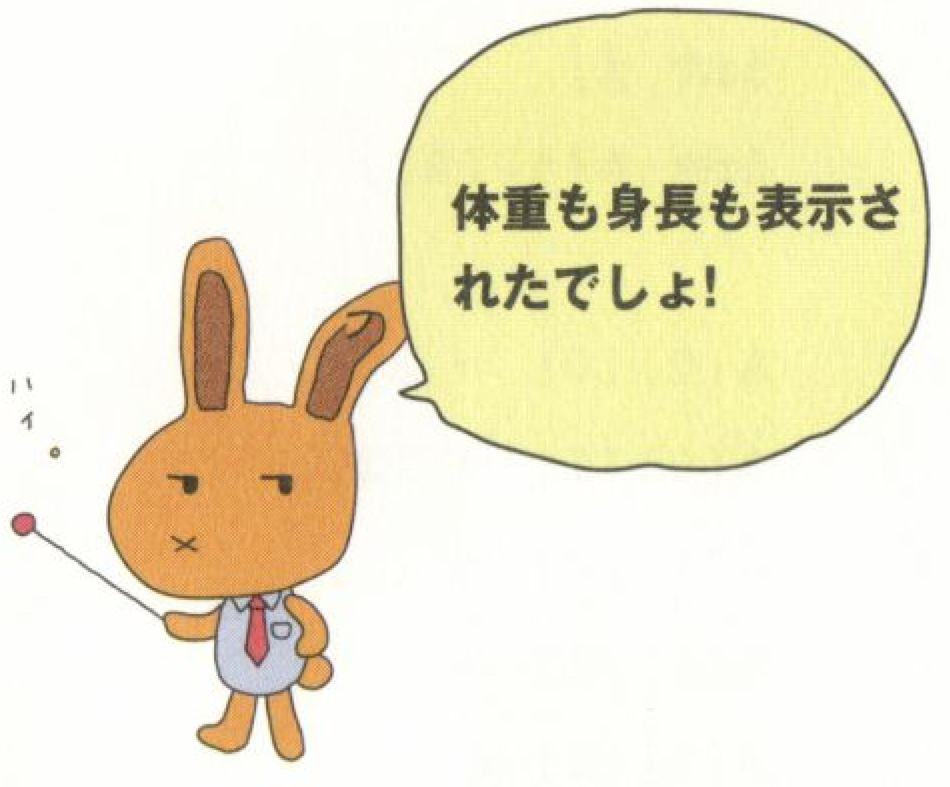
❸ コマンドプロンプトを起動します。

❹「**cd c:¥MyC** Enter」と入力してください。

❺「**gcc MyC112.c -o MyC112** Enter」と入力します。

❻「**MyC112** Enter」と入力します。

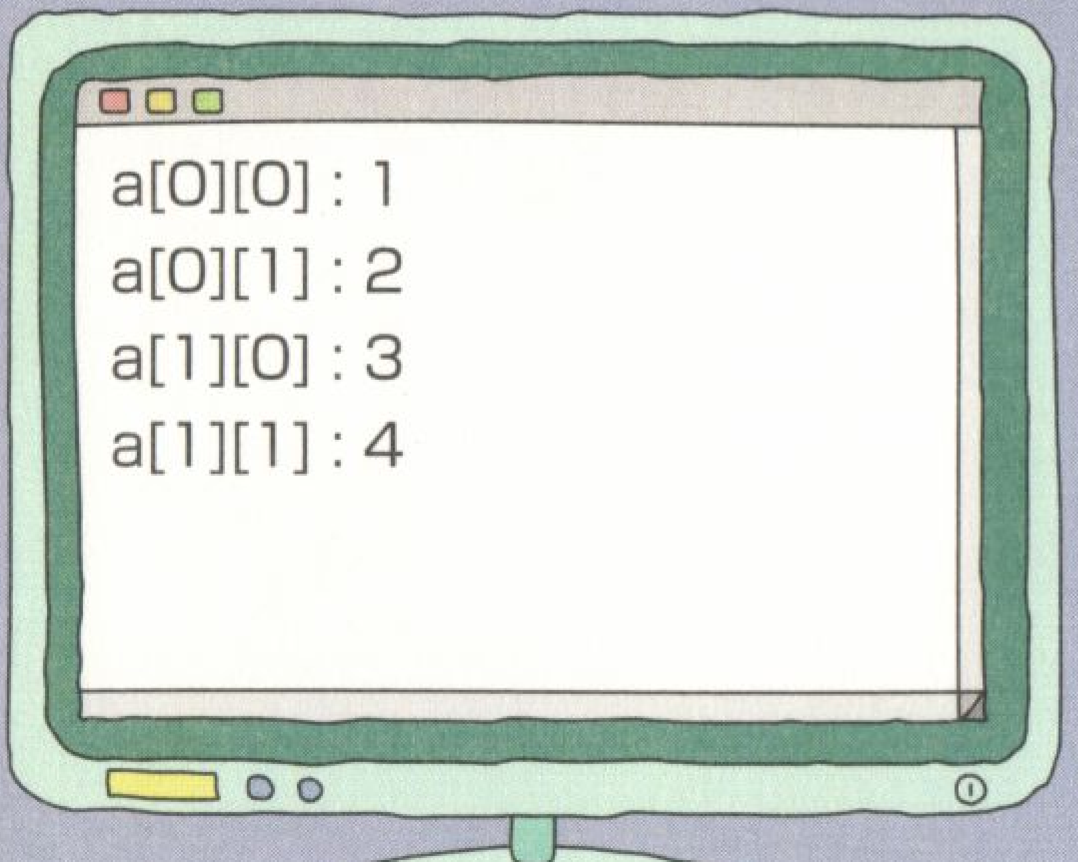




配列をこのように使えば、体重だけでなく、身長も表示することができるようになります。

ところで、さらに何かを表示しなければならなくなっても、心配はいりません。配列の列を2から3へ、3から4へというように増やしていくことができます。このように、列が1より大きい配列（ここで扱った2次元配列も含めて）を**多次元配列**といいます。

例題　多次元配列



次のように画面に表示するソースコードを、int型2次元配列と、2重ループ（繰り返しの中で繰り返しを使うこと）を使って書いてください。

●解答

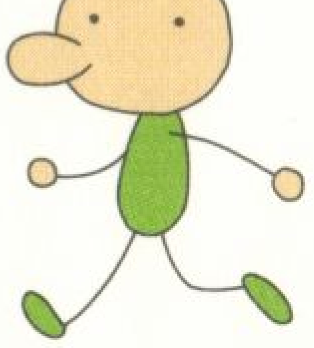
```c
/* Exercise141.c */
#include <stdio.h>

int main(void)
{
  int i;
  int j;
  int a[2][2];

  a[0][0] = 1;
  a[0][1] = 2;
  a[1][0] = 3;
  a[1][1] = 4;

  for (i = 0; i < 2; i++) {
          for (j = 0; j < 2; j++) {
                  printf("a[%d][%d] : %d¥n", i, j, ➡
                                                ➡ a[i][j]);
          }
  }

  return 0;
}
```



※注意　ソースコード内の➡は、レイアウトの都合により行を折り返していますが、実際には1行で入力してください。

practice 練習問題

解答は213ページ

1 条件分岐 難易度★★★☆☆

int型の変数aに値を代入して、7以上から12以下であれば、「elementary school」、13以上15以下なら「junior high school」、それ以外は「unknown」、と画面に表示するプログラムを書いてください。

▶▶ヒント：本文99ページ参照

2 繰り返し 難易度★★★★★

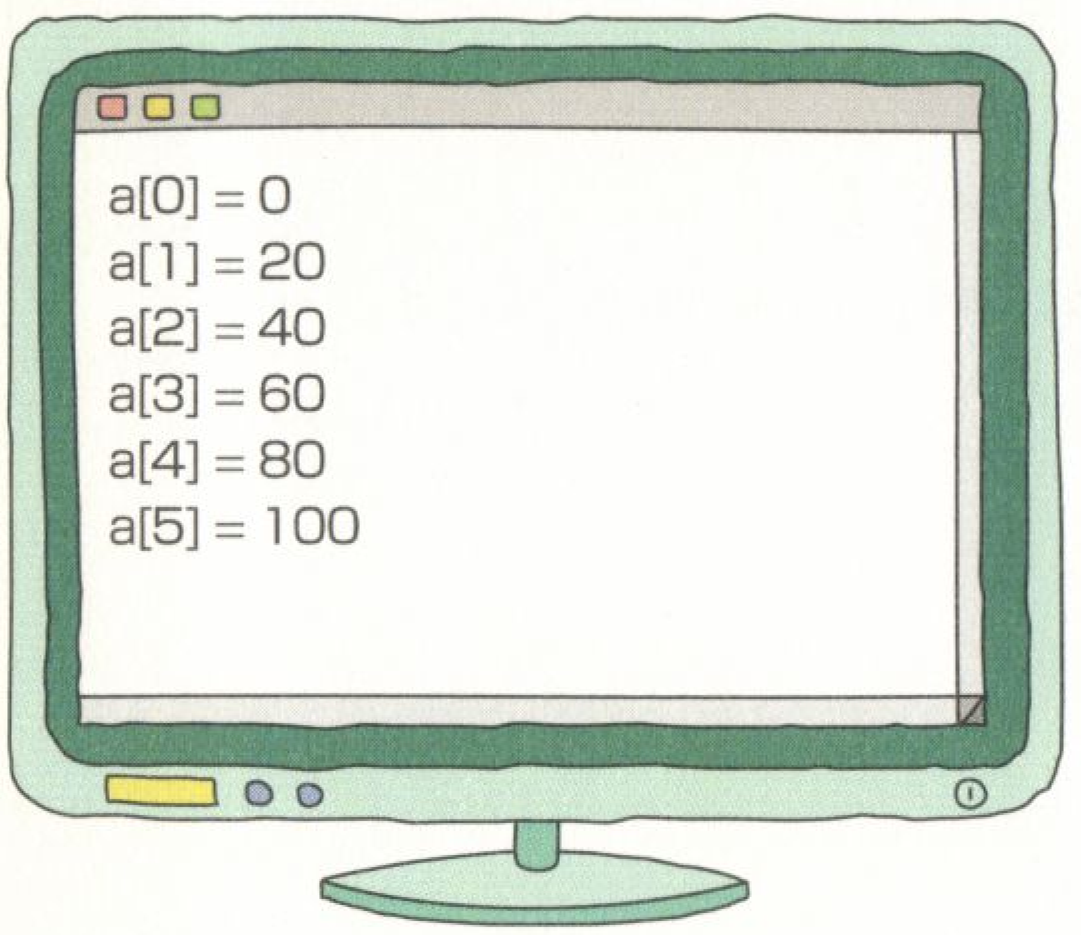
九九のプログラム「My C8 . c」を全段完成させてください。

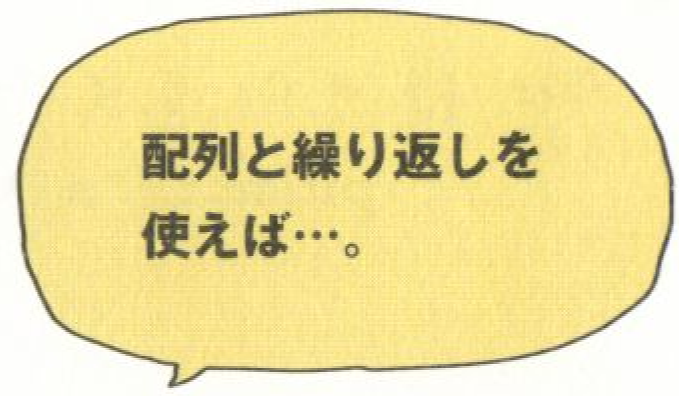
▶▶ヒント：本文114ページ参照

3 繰り返し 難易度★★★☆☆

次のように、画面に出力するソースコードをint型の配列a[]と繰り返しを使って書いてください。

▶▶ヒント：本文114ページ参照





memo

Chapter5のまとめ

★制御文

制御文とは、プログラムの実行の流れを変える文のことで、条件分岐と繰り返しがあります。

★制御構造

制御構造とは、プログラムの流れを決定するものです。すべてのアルゴリズムは、3つの制御構造（順次実行、条件分岐、繰り返し）の組み合わせで表せます。

★条件分岐

条件分岐とは、条件によって実行する文を変更することです。

★関係演算子

関係演算子とは、値の大きさを比較する演算子です。

★論理演算子

論理演算子とは、複雑な条件を作るときに使います。

★繰り返し

繰り返しとは、同じ文を何回も実行することです。

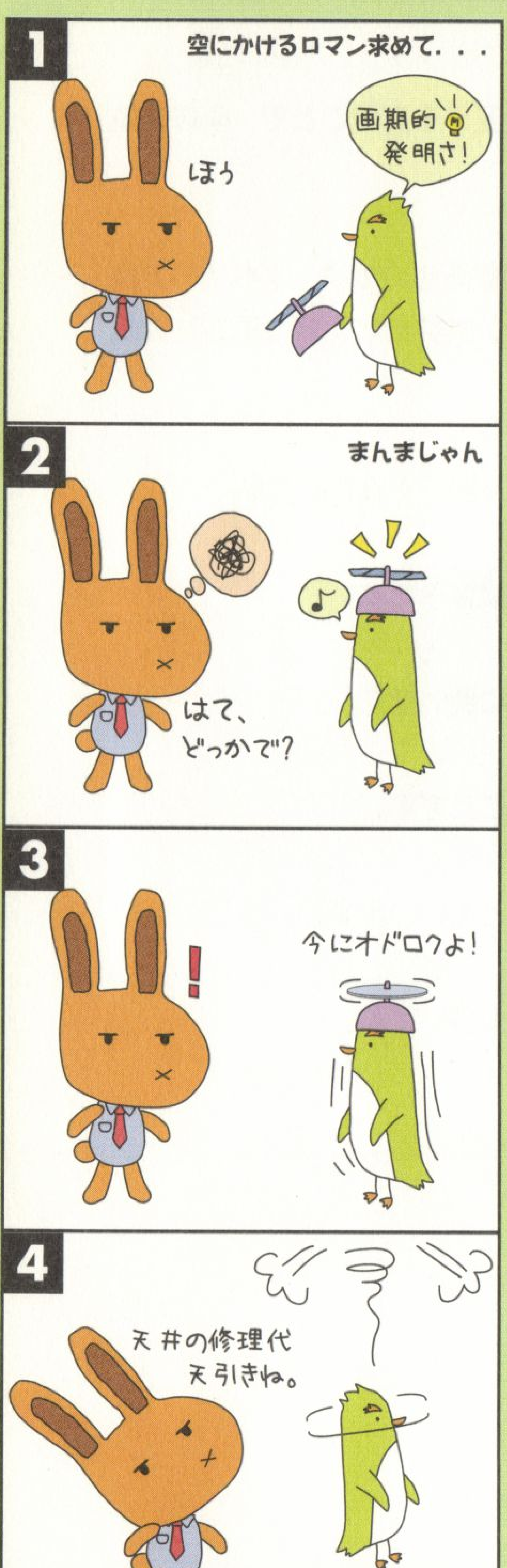
★配列

配列とは、組になった同じ型の変数のことをいいます。

★繰り返しと配列

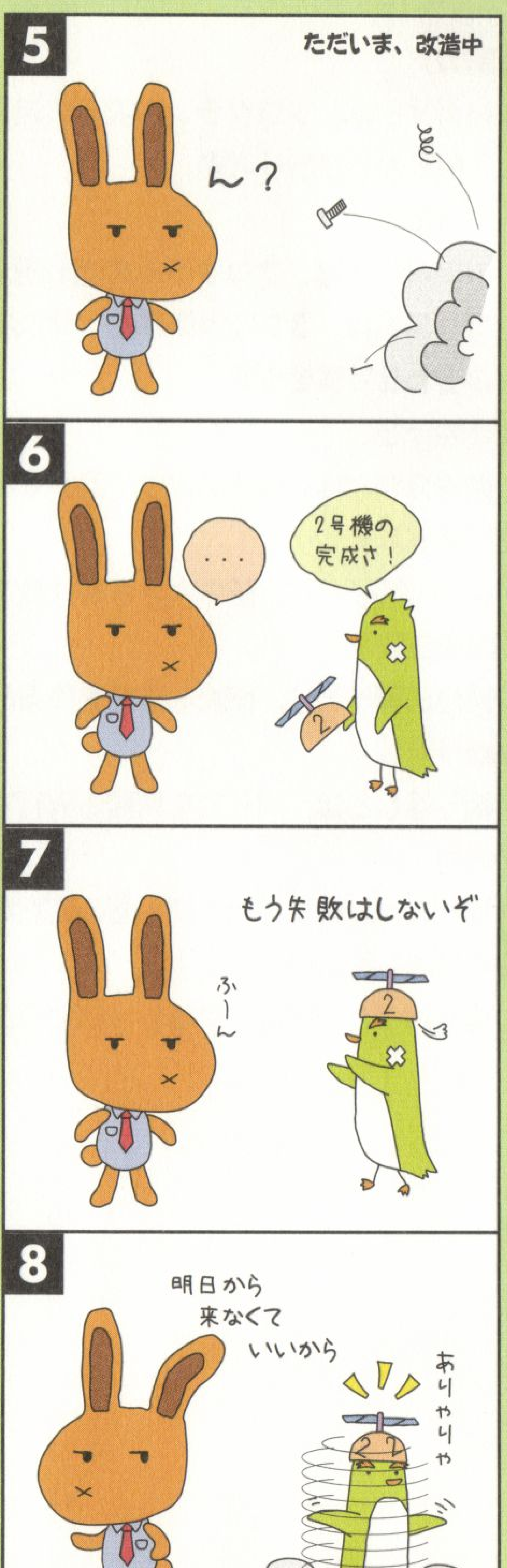
繰り返しと配列は、いっしょに使うことが多いです。

大発明!!

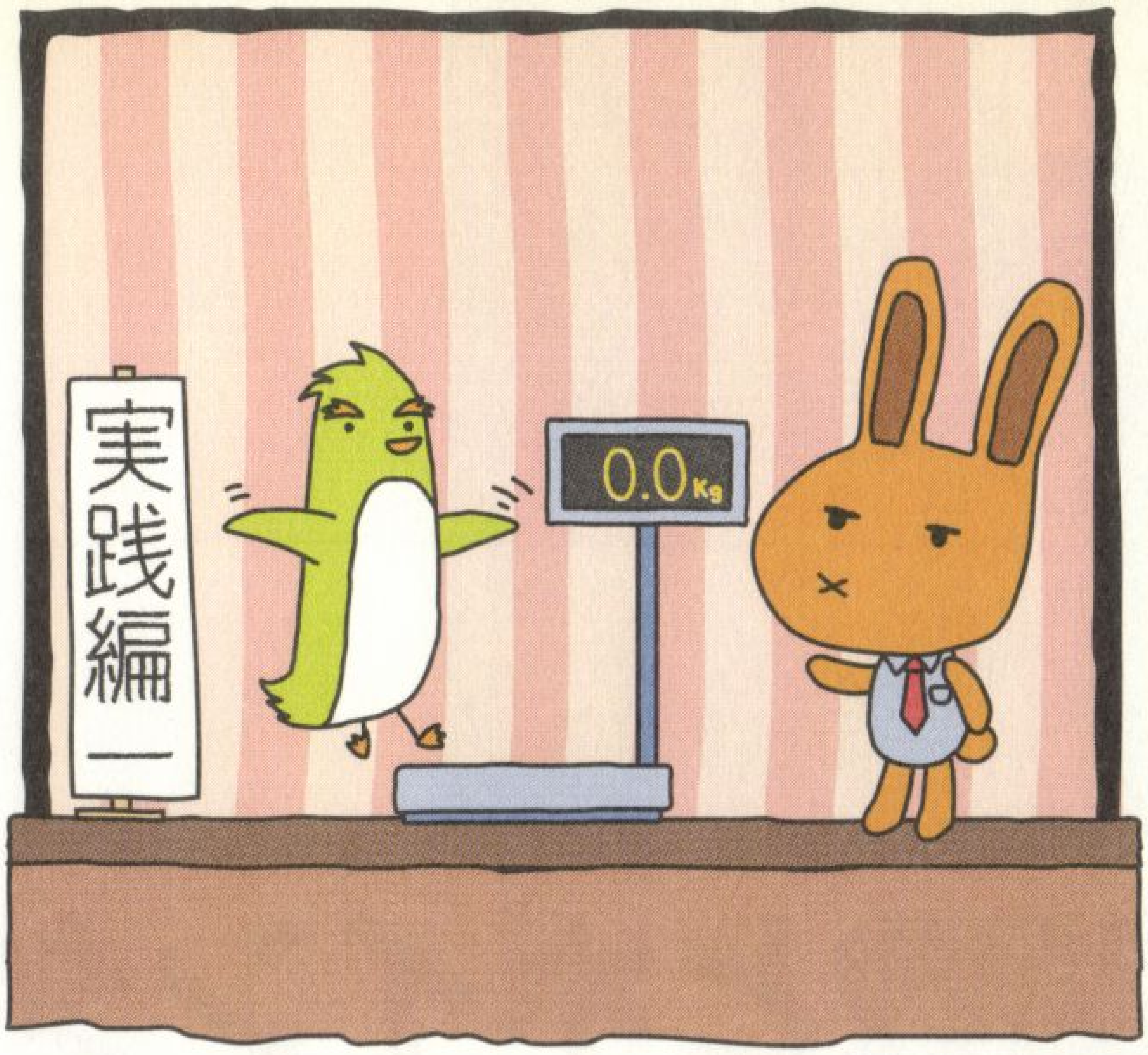


空にかけるロマン求めて. . .
ほぅ
画期的発明さ!
まんまじゃん
はて、どっかで?
今にオドロクよ!
天井の修理代天引きね。

続・大発明!!



ただいま、改造中
ん?
2号機の完成さ!
もう失敗はしないぞ
ふーん
明日から来なくていいから
ありゃりゃ



★いままで学んだ知識を、より高度に発展させます。
ここでは、関数とポインタについて学んでいきます。

ここがC言語プログラミングの正念場です。

★C言語プログラミングの根本理解と
全体のまとめを学びたい方は

➡ 第7章本文168ページへ

★C言語プログラミングの応用を確認したい方は

➡ 第4章本文72ページへ

Chapter 6

関数とポインタ



この章でマスターすること

この章では、はじめに**関数**について説明します。

文を作るための規則やプログラミング言語独特の考え方がわかれば、文が作れますね。コンピュータにうまく指示するために、この文をどうやって組み合わせればよいかを考えるのが、プログラミングです。

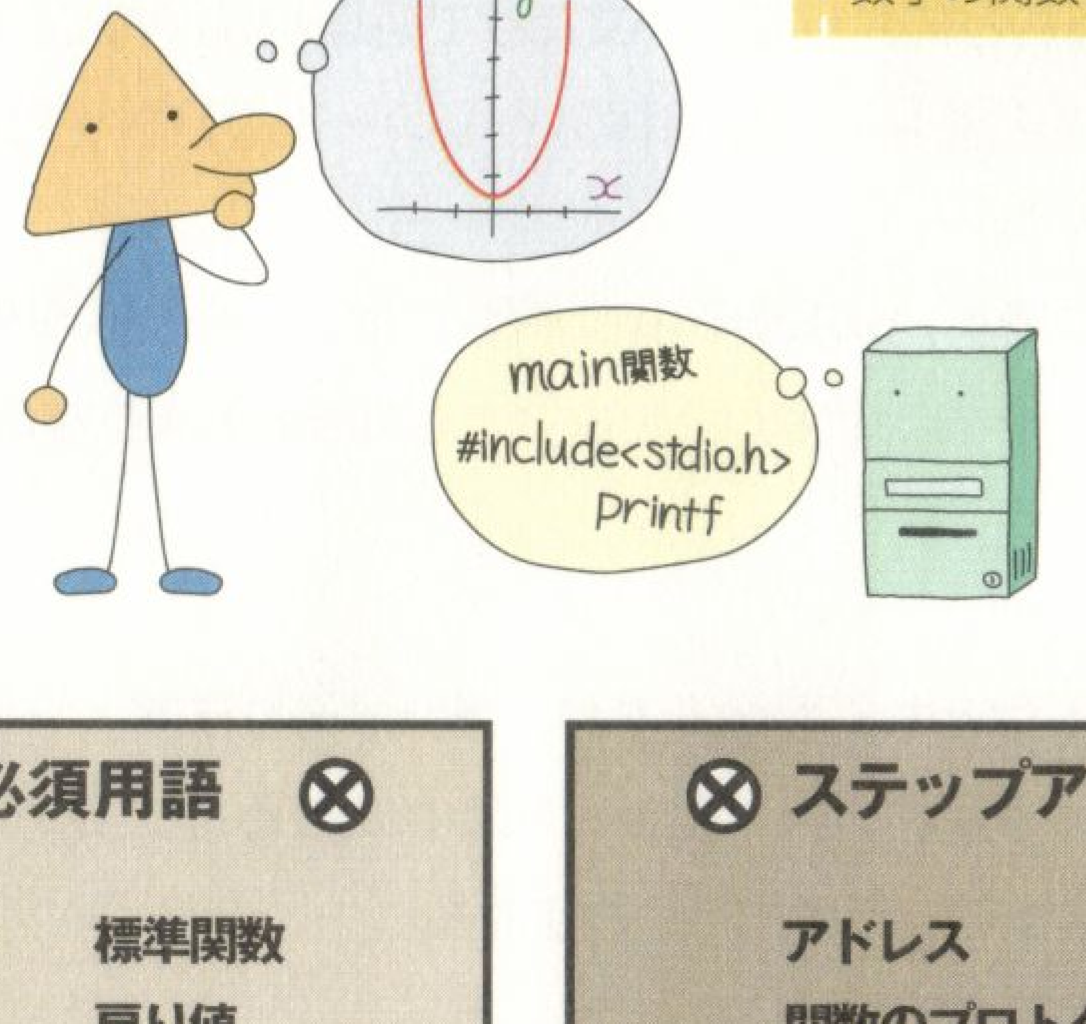
ある処理に関係のある「文」をまとめたものを、関数といいます。C言語のソースコードは、main関数も含めて関数の集まりです。

次に、**アドレス**と**ポインタ**について説明します。変数は、コンピュータの記憶領域（メモリ）に確保されるデータを、記憶するための場所のことでした。この場所の具体的な位置のことを「アドレス」といいます。「ポインタ」は、この「アドレス」と関係があります。

C言語には、ポインタを使うことができるという特徴がありますが、ポインタは、コンピュータのハードウェアと密接に関わりますので、実際に使いこなすことは、いまの段階では難しいと思います。ポインタとは、こういうものなのだと頭の片隅にでも置いてください。

それでは、関数とポインタについて学んでいきましょう。

数学の関数とどう違う？



C言語必須用語

関数　標準関数
引数　戻り値

ステップアップ用語

アドレス　ポインタ
関数のプロトタイプ宣言

Section 01 関数

「関数」とは何のことをいうのでしょうか？　数学でも関数を習いますが、C言語のものとはちょっと違います。ここでは、C言語プログラミングにおける「関数」の仕組みを学んで作ってみましょう。

関数

C言語でいう**関数**とは、与えられた情報に従って、何らかの処理を行ってくれて何度でも簡単に利用できるものです。数学の関数とは少し違います。

C言語のプログラムは、この「関数の集まり」でできているのです。そして関数とは、自分が作るものと、もともと用意されているものがあります。 つまり、自分が作った関数と、もともと設定されている関数を組み合わせることで、プログラムを完成させるのです。

いままで、様々なC言語のソースコードを見たり、書いたりしてきましたが、その中で結果を画面に出力するとき、必ず次のような文があったと思います。

```
printf(・・・);
```

このprintfとは何でしょうか？

画面に文字や変数の値を出力させるには、本来は様々な処理をしなければなりません。しかし、「printf」文があることによって、一文だけで処理がなされているのです。

このように、「何かをするためにしなければならない一連の処理をまとめて、名前を付けたもの」を、関数といいます。

main関数も、関数の一つです。いままでmain関数には、一連の処理を書いてきましたね。

実は、C言語のソースコードは、main関数も含めて関数の集まりなのです。

▶専門的な説明

「main関数」と「printf」はともに関数ですが、違いがあります。

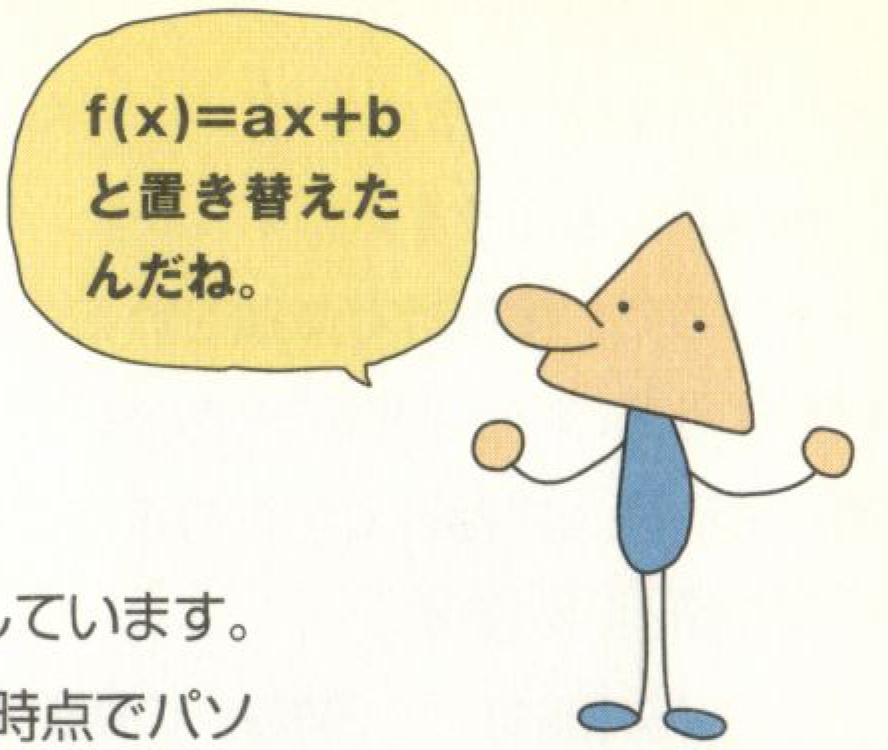
「main関数」の中身は、実際にソースコードの中にあるので見ればわかるのですが、「printf」は、中身がどうなっているかがわかりません。

これは、次のように考えてください。

ここに次のような式があります。

$Y = ax + b$

これは、数学でいう「YはXの関数であり、Xの値が決まると、右辺よりYの値が決まる」という関数※を表しています。そこで、この式をf(x)という記号を使って書き直して、Y=f(x)とします。つまりこれは、「ax+bという処理をまとめて、f(x)という記号で、この関数を表現した」わけですね。



main関数は「ax + b」に、「printf」は f(x) に対応しています。

実際は、「printf」の中身はgccをインストールした時点でパソコン上のどこかにありますが、その中身を見る機会はあまりありません。「printf」は、カッコの中に示した文字や変数の値を画面に出してくれさえすればいいので、それ以上深くは考えないからです。そして、「printf」を使えば必ず画面上に文字が表示されることは、約束されているものとして使っています。

このような関数は、C言語には多数用意されていて、そのことを**標準関数**といいます。この用意されている関数を使うために、「**#include <stdio.h>**」というおまじないを使っていたのです。「#include <stdio.h>」の意味は、「『printf』が用意されている、『stdio.h』というヘッダーファイルを、これから使いますよ」ということなのです。

C言語には、様々な分野の標準関数が用意されていますが、よく使うものには、次のものがあります（標準関数のもう少し詳しいことは、あとで説明します）。

標準関数	文字列処理関数
	数学関数
	入出力関数

関数の仕組み

もう少し、関数について考えてみましょう。関数とは、「何かをするためにしなければならない一連の処理をまとめて、名前を付けたもの」だと説明しました。実は、この関数に相当するものは、生活の中にもあるのです。

例えば、体重計を考えてください。

体重計で体重を計るとき、まず体重計に乗ります。そうすると、指針が目盛を指します（体重を教えてくれます）。

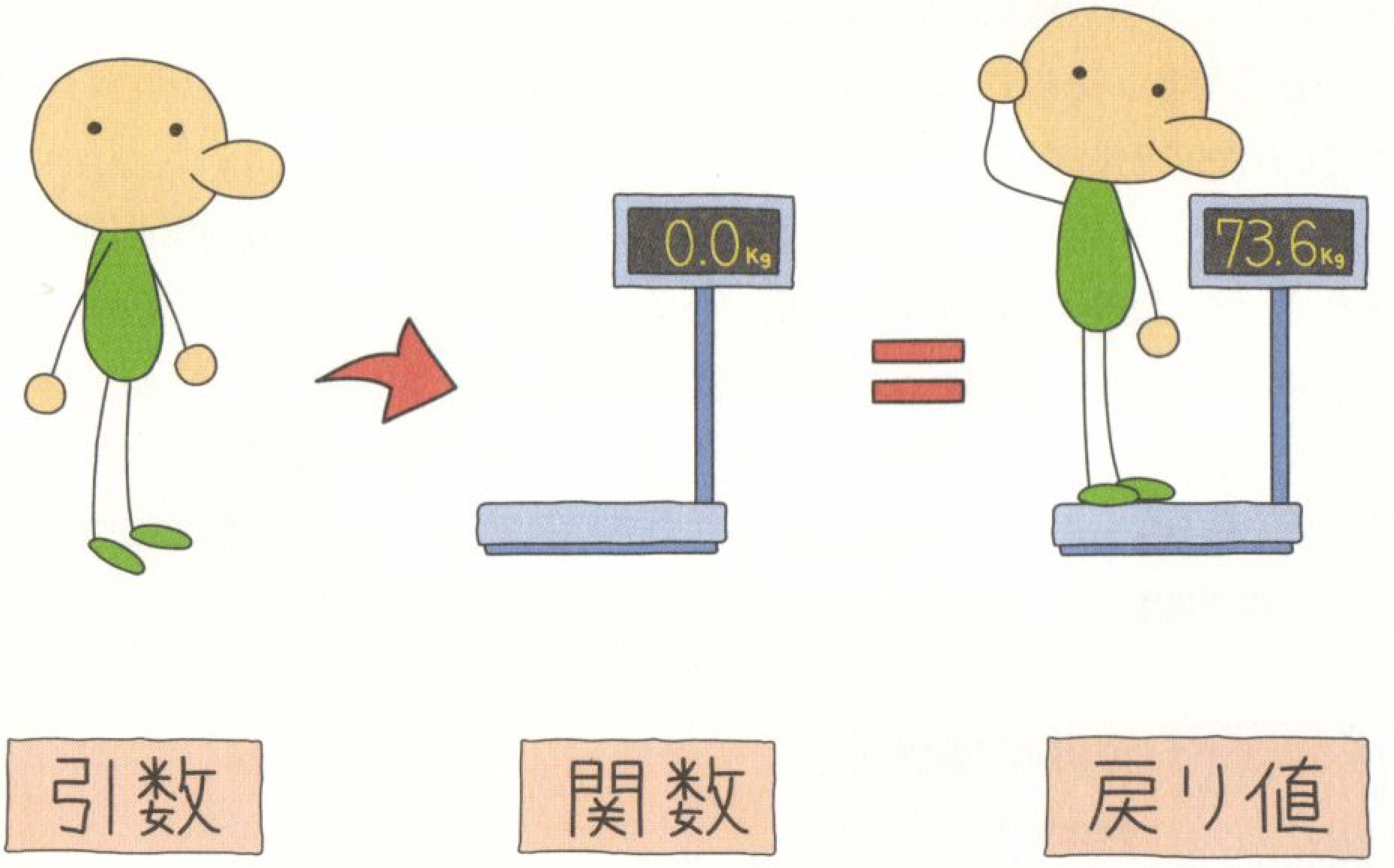
このとき体重計は、「体重計に乗った人の体重を計って、その人の体重に相当する目盛に指針を動かす」という一連の作業をしてくれますね。つまり、体重計は、関数と同じ働き

※数学でいう関数　2つの変数（X,Y）において、Xの値が一つ決まると、それに応じてYの値が一つ決まる関数のこと。「yはxの関数である」。

をしてくれているのです。しかも、ふつうは体重計の中身は見られないので、まるで「printf」のようです。

ところで、体重計で体重を計るとき、「人」がその上に乗りました。実は、この「人」に相当するものが、関数にもあるのです。そのことを、**引数**といいます。また、計った「体重」は目盛上に指針で示されます。この「体重」に相当するものも、関数にはあり、それを**戻り値**といいます。

この「体重計」と「関数」の関係を、イラストで示してみましょう。



▶▶関数を作る

ここまで、関数について説明してきたのですが、実際に作成してみることで、関数がどういうものなのかを実感できると思います。それでは、C言語で関数を作成してみましょう。

まず、次のソースコードをはじめに書いてみます。

```c
/* MyC121.c */
#include <stdio.h>

int main(void)
{
  int a;
  int b;
  int c;

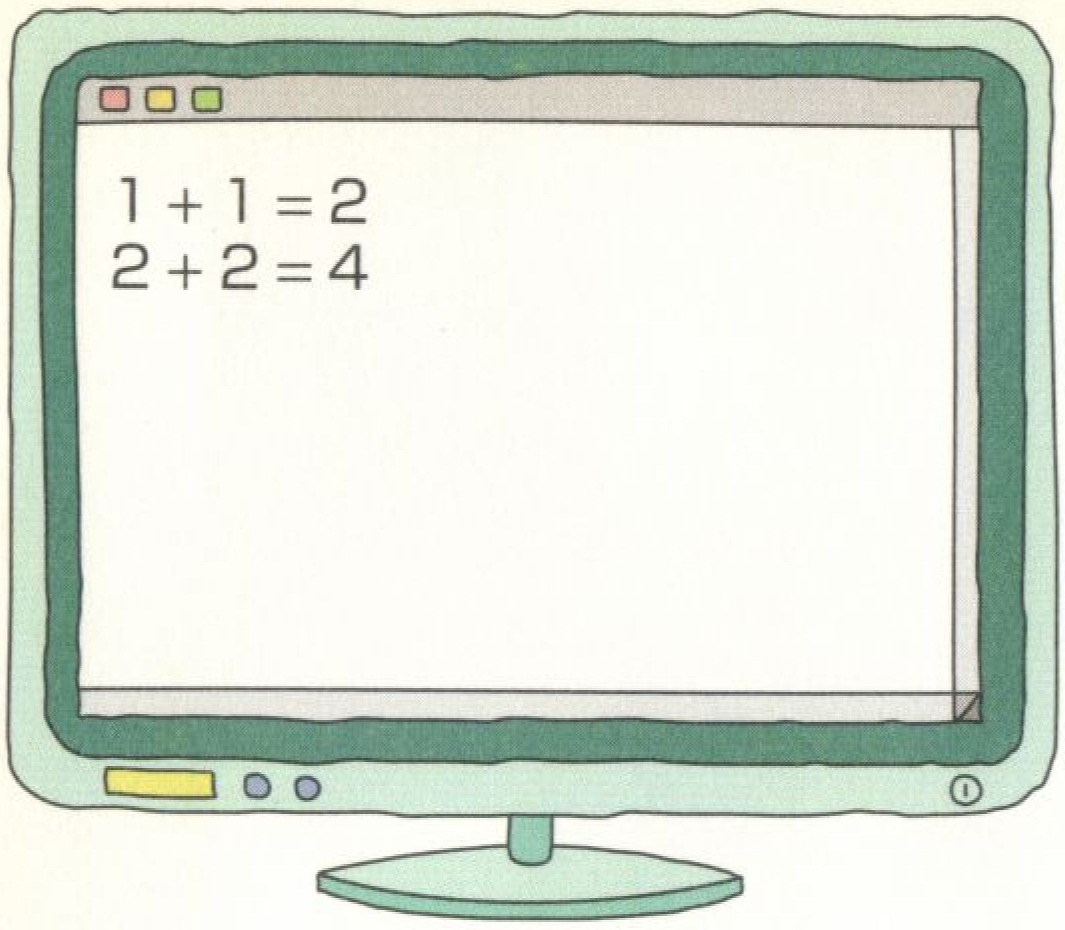
  a = 1;
  b = 1;
  c = a + b;
  printf("%d + %d = %d¥n", a, b, c);

  a = 2;
  b = 2;
  c = a + b;
  printf("%d + %d = %d¥n", a, b, c);

  return 0;
}
```


▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷ 「MyC」フォルダにファイル名「MyC121.C」で保存します。
❸ コマンドプロンプトを起動します。
❹ 「**cd c:¥MyC** Enter」と入力してください。
❺ 「**gcc MyC121.c -o MyC121** Enter」と入力します。
❻ 「**MyC121** Enter」と入力します。



12行目と17行目は変数に代入されている値が違うだけで、たし算をするという部分はまったく同じ処理ですね。この部分を関数にしてやりましょう。関数は、決まった書式に則って作成するのですが、ここではとりあえず先に関数を作成してみます。

次の文は「関数として、たし算部分を一連の処理としてまとめたもの」、すなわち関数「Tashizan」です。

関数の書式は、次のようになっています。

```
int Tashizan(int a, int b)
{
  int c;

  c = a + b;

  return c;
}
```

書式

```
戻り値の型 関数名(引数の型 引数名)
{
        文1;
        文2;
         :

        return 戻り値; ──▶ 戻り値の型が「void」ならば、必要ありません
}
```

それでは、関数の「Tashizan」の書式を使って見てみましょう。

```
/* MyC122.c */
#include <stdio.h>

int Tashizan(int a, int b) ──→戻り値の型は、int型
{
  int c; ──→int型変数「c」の宣言

  c = a + b;

  return c; ──→戻り値（変数「c」の値）は、int型
}

int main(void)
{
  int a;
  int b;
  int c;

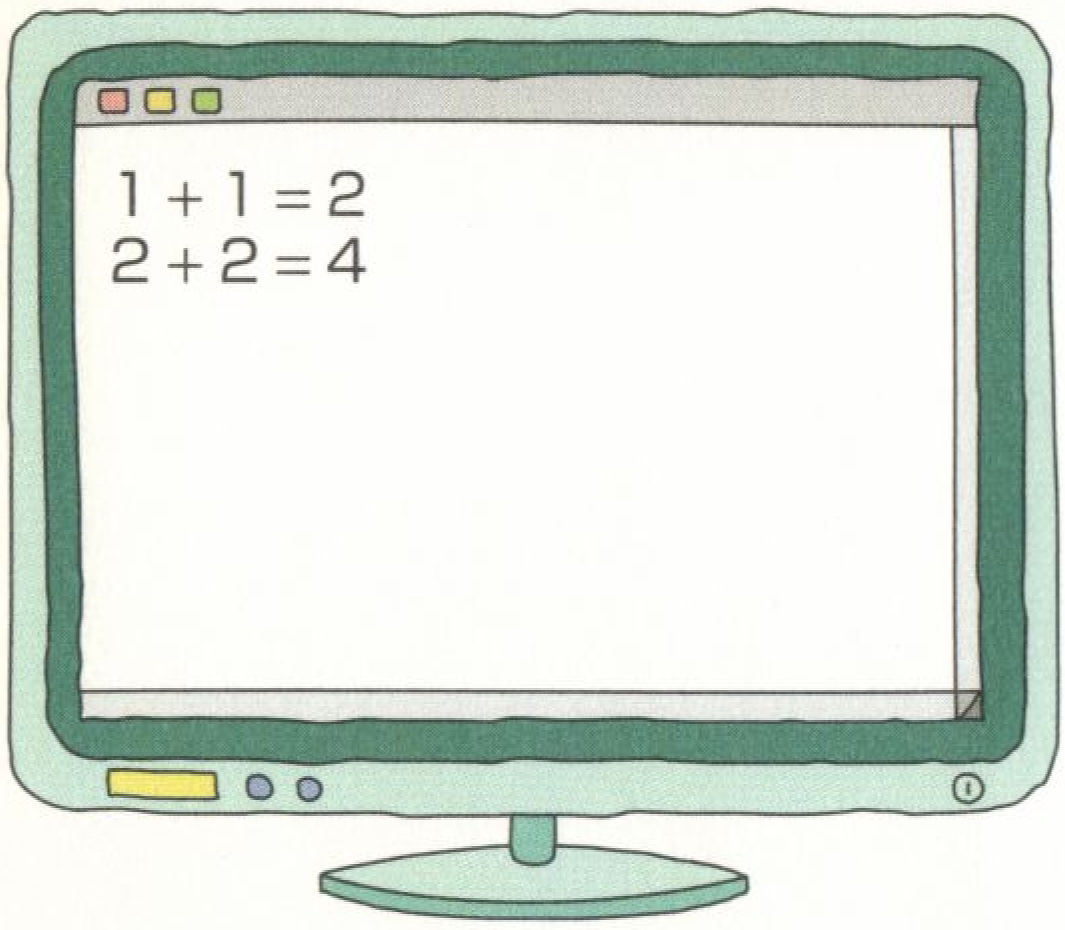
  a = 1;
  b = 1;
  c = Tashizan(a, b); ──→引数「a、b」を関数「Tashizan」に渡して、戻り値を変数「c」に代入
  printf("%d + %d = %d¥n", a, b, c);

  a = 2;
  b = 2;
  c = Tashizan(a, b); ──→引数「a、b」を関数「Tashizan」に渡して、戻り値を変数「c」に代入
  printf("%d + %d = %d¥n", a, b, c);

  return 0;
}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC122.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC122.c -o MyC122** Enter」と入力します。
❻「**MyC122** Enter」と入力します。





関数は、原則的にはmain関数の前に書きますので、4行目から11行目に関数「Tashizan」を書いてあります。関数「Tashizan」は、戻り値である変数「c」の値がint型なので、4行目の文頭は「int」になります。

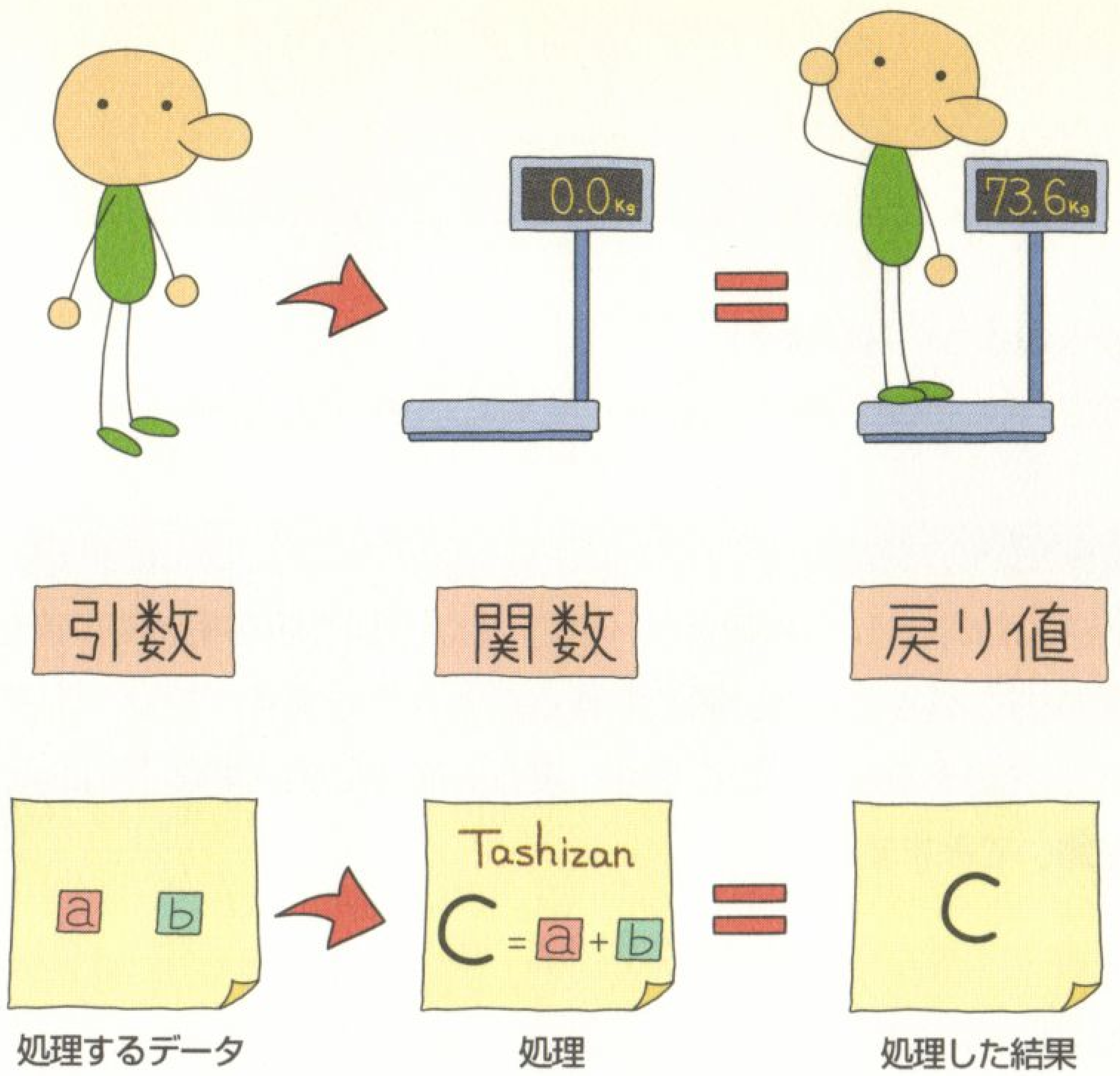
▶専門的な説明

関数は、原則的にはmain関数の前に書くのですが、関数「printf」は書いてありませんね。関数「printf」は、どこに書いてあるのでしょう？

それは、「#include <stdio.h>」の中にあると考えてください（本当は、ちょっと違うのですが、いまはこう思っていてかまいません）。

いままで、気にせず書いてきた「#include <stdio.h>」ですが、実は、このために必要だったのです。

21行目と26行目で関数「Tashizan」を使っています。ここでは、引数「a、b」を関数「Tashizan」に渡して、戻り値を変数「c」に代入しています。先ほど関数を説明するために、体重計を例に出したのですが、関数「Tashizan」を体重計の働きと関連付けて、もう一度見てみたいと思います。



　また、前に「プログラムデータは、main関数の上から下に向かって、ソースコードに書かれた順番に実行されていきます」と書いたのですが、main関数とその他の関数は、次のように実行されていきます。

書式

```
#include<stdio.h>
関数A
{
    文A;
    文B;
    ○○○
}

main関数
{
    文1;
    文2;
    関数A;
    文3;
```

```
      ⋮
}
```

実行順序は以下のようになります。

「文1」➡「文2」➡「文A」➡「文B」➡「○○○」➡「文3」➡「・・・」

▶▶関数いろいろ

さて、体重計は、「人」という引数と、「体重」という戻り値を持っていましたが、C言語では、引数や戻り値を必要としない関数も作ることができます。サンプルプログラムソース「MyC122.c」を書き直して、ここでは、戻り値を持たない関数「Tashizan」を使ったソースコードを書いてみます。

```
/* MyC123.c */
#include <stdio.h>

void Tashizan(int a, int b)
{
  int c;

  c = a + b;
  printf("%d + %d = %d¥n", a, b, c);
}

int main(void)
{
  int a;
  int b;

  a = 1;
  b = 1;
  Tashizan(a, b);

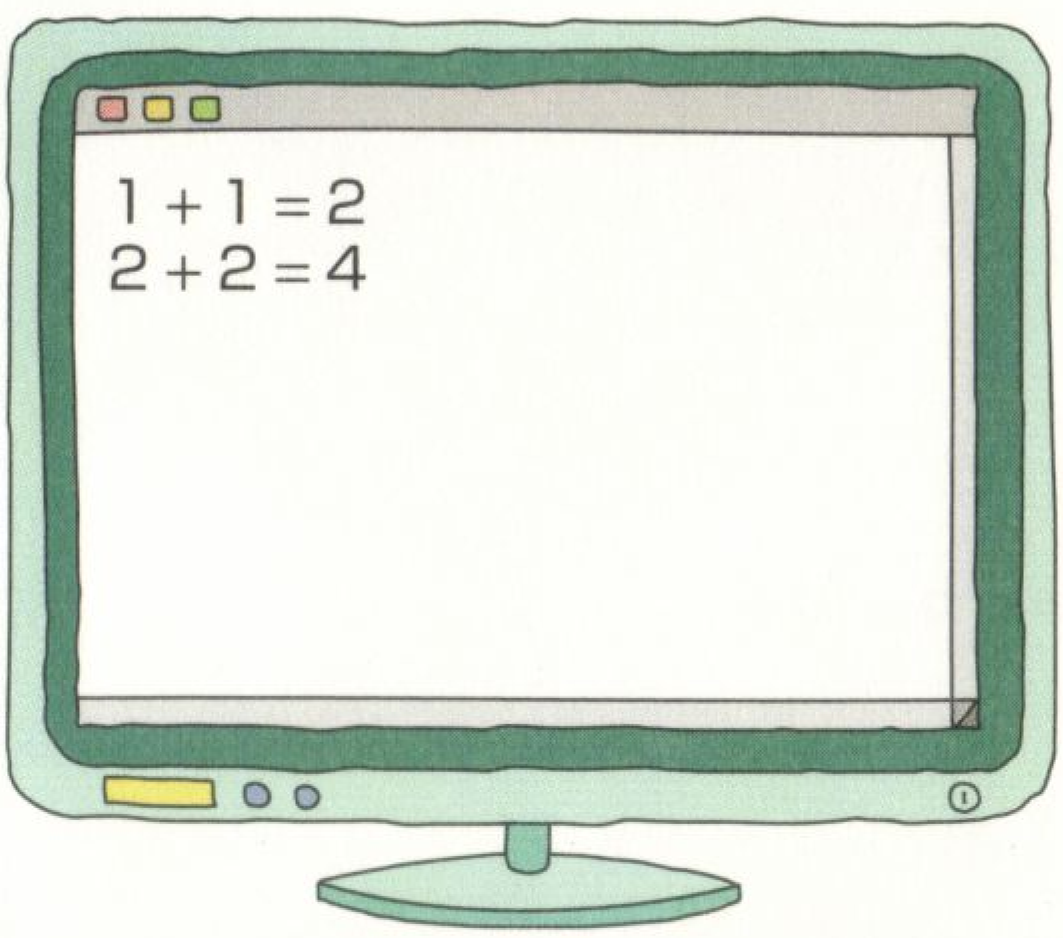
```

関数が戻り値を持たないとき、「void」となります

```
21:   a = 2;
22:   b = 2;
23:   Tashizan(a, b);
24:
25:   return 0;
26:}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC123.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC123.c -o MyC123** Enter」と入力します。
❻「**MyC123** Enter」と入力します。





「MyC122.c」との違いがわかりますか？

4行目から10行目を見てください。「MyC122.c」の関数「Tashizan」とは違って、「MyC123.c」の関数「Tashizan」は戻り値がありませんね（4行目の「void」は、関数が戻り値を持たないときに、この位置に書きます）。そのため、19行と23行で関数「Tashizan」を使うときにも戻り値を代入する必要がありません。

ただし、計算結果を表示する値がなくなってしまったので、9行目に結果を表示する文を移動しました。「MyC122.c」では、結果を表示する部分も2回繰り返していました（22行目と27行目）ので、結果としては、こちらのほうが「処理をまとめる」関数らしくなったようです。

▶専門的な説明

ところで、「MyC123.c」を次のようにしてはいけないのでしょうか？

```
/* MyC124.c */
#include <stdio.h>

void Tashizan(int a, int b)
{
  int c;

  c = a + b;
}

int main(void)
{
  int a;
  int b;

  a = 1;
  b = 1;
  Tashizan(a, b);
  printf("%d + %d = %d¥n", a, b, c);

  a = 2;
  b = 2;
  Tashizan(a, b);
  printf("%d + %d = %d¥n", a, b, c);

  return 0;
}
```

19行目と24行目で、関数「Tashizan」で宣言した変数「c」を使っていますが、残念ながらmain関数の中で、この変数「c」は無効なのです。つまり、関数の中で変数を使うときは、その変数（関数の中で宣言した変数のことを**ローカル変数**といいます）は、その関数の中でしか使えないのです。言い換えると、C言語では関数が違えば同じ変数名を使用してもかまわないのです。ただし、その関数の内部でのみ、この変数は有効であるということに注意してください。

ちなみに、このソースコードを実行させてみてください。次のようになります。

```
C:¥MyC>gcc MyC124.c -o MyC124
MyC124.c:  In function `main':
MyC124.c:19:  `c'undeclared(first use in this function)
MyC124.c:19:  (Each undeclared indentifier is reported
only once
MyC124.c:19:  for each function it appears in.)

C:¥Myc>
```

例題 関数

本文143ページのサンプルソースコード「MyC121.c」と同じ結果を出力するソースコードを、引数も戻り値も持たない関数「Tashizan」で書いてください。

●解答

```
/* Exercise14.c */
#include <stdio.h>

void Tashizan(void)
{
  int a;
```

関数が引数を持たないとき、「()」の中は「void」となります

```
 7:  int b;
 8:  int c;
 9:
10:  a = 1;
11:  b = 1;
12:  c = a + b;
13:  printf("%d + %d = %d¥n", a, b, c);
14:  a = 2;
15:  b = 2;
16:  c = a + b;
17:  printf("%d + %d = %d¥n", a, b, c);
18:}
19:
20:int main(void)
21:{
22:  Tashizan();
23:
24:  return 0;
25:}
```

●解説

4行目を見てください。関数が引数を持たないとき、「()」の中は「void」となります。また、22行目で引数も戻り値も持たない関数「Tashizan」を使っています。

ところで、引数も戻り値も持たない関数「Tashizan」を作成したことで、いままでmain関数の中にあった処理のほとんどが「Tashizan」に移動してしまいました。こうなってしまうと、どちらがメインの関数なのかわからなくなってしまいますね。ここではサンプルとして書いたのですが、実際のプログラミングでは、このような関数を作成してもあまり意味がありません。

いままで、「MyC122.c」、「MyC123.c」、「Exercise14.c」という関数を使うソースコードを見てきたのですが、最もうまく関数を使ったものは「MyC123.c」です。これは、「一連の処理をまとめる」という関数の考えに最も沿っているからです。

例題 関数❷

本文148ページのサンプルソースコード「MyC123.c」を参考にして、関数「Tashizan」以外に、引き算をする関数「Hikizan」を作成して、その結果を出力するソースコードを書いてください。

●出力例

1＋1＝2
2－1＝1

●解答

```
/* Exercise15.c */
#include <stdio.h>

int Tashizan(int a, int b)
{
  int c;

  c = a + b;
  printf("%d + %d = %d¥n", a, b, c);
}

int Hikizan(int a, int b)
{
  int c;

  c = a - b;
  printf("%d - %d = %d¥n", a, b, c);
}

```

```
int main(void)
{
  int a;
  int b;

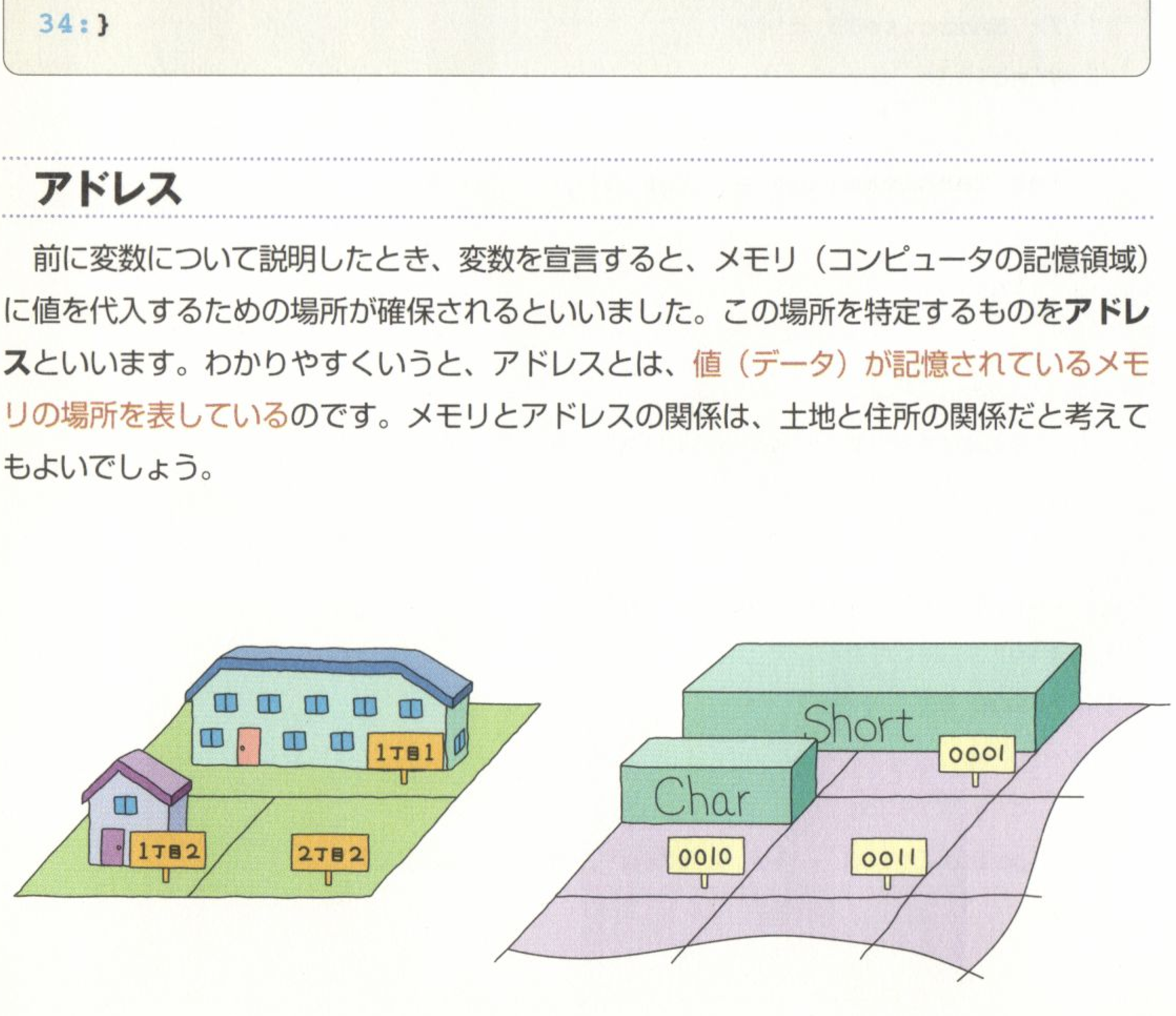
  a = 1;
  b = 1;
  Tashizan(a, b);

  a = 2;
  b = 1;
  Hikizan(a, b);

  return 0;
}
```

アドレス

前に変数について説明したとき、変数を宣言すると、メモリ（コンピュータの記憶領域）に値を代入するための場所が確保されるといいました。この場所を特定するものを**アドレス**といいます。わかりやすくいうと、アドレスとは、値（データ）が記憶されているメモリの場所を表しているのです。メモリとアドレスの関係は、土地と住所の関係だと考えてもよいでしょう。



実際に、アドレスがどういうものなのかを実感するために、ここでは、アドレスを表示するソースコードを見てみましょう。

```
/* MyC13.c */
#include <stdio.h>

int main(void)
{
  int a;

  a = 1;

  printf("a: value[%d],Address[%p]¥n", a, &a);

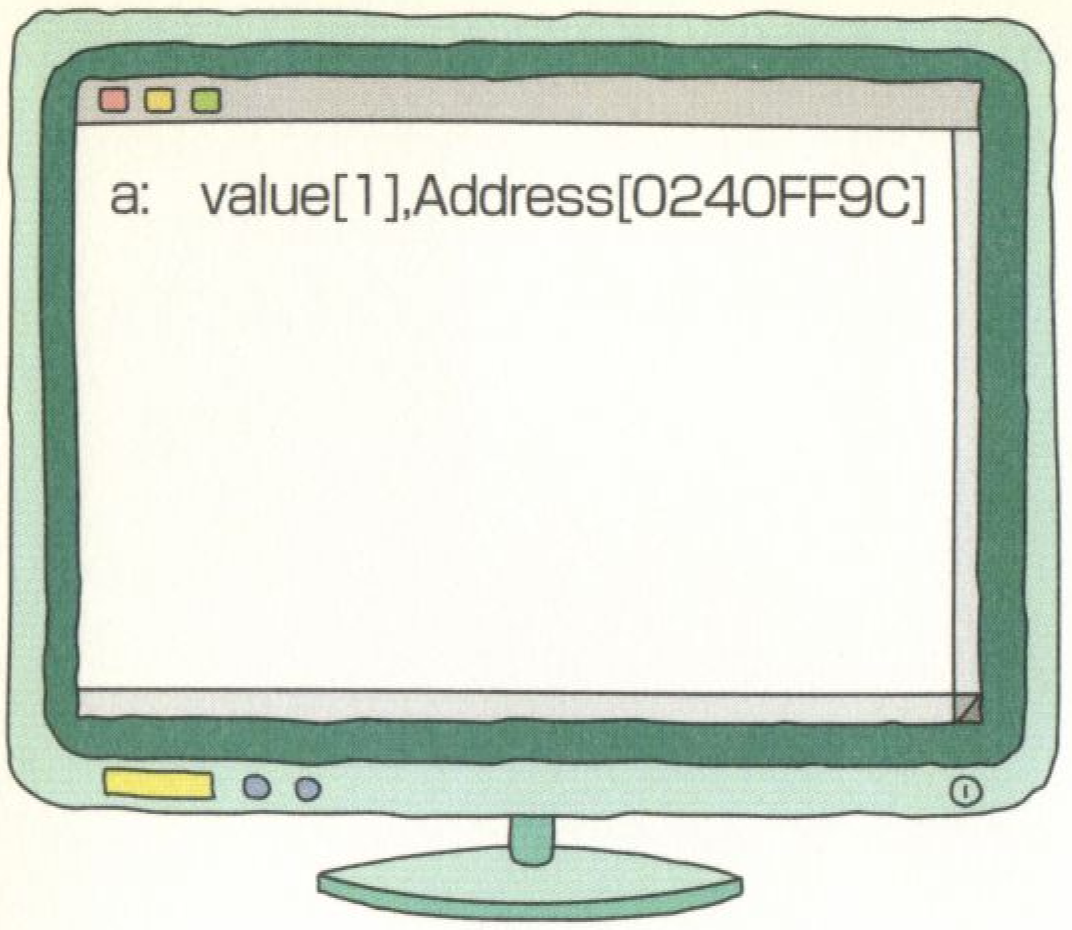
  return 0;
}
```

「&a」の「&」は、アドレスを取り出すための記号
「%p」は、アドレスを表示するときに使う

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷ 「MyC」フォルダにファイル名「MyC13.c」で保存します。
❸ コマンドプロンプトを起動します。
❹ 「`cd c:¥MyC` Enter」と入力してください。
❺ 「`gcc MyC13.c -o MyC13` Enter」と入力します。
❻ 「`MyC13` Enter」と入力します。







10行目を見てください。「&a」の「&」は、アドレスを取り出すための記号で、「%p」は、アドレスを画面に表示するときに表示したい場所に埋め込みます。

このソースコードを実行すると、int型の変数「a」の値とアドレスが表示されます。

変数aの値は、このアドレスに確保されているわけです。

▶専門的な説明

長いソースコードを書いているときなど、関数を使うときに引数の型や個数を間違ってしまうときもあります。そこでC言語には、関数を使う前に関数の引数の型などをチェックする仕組みがあります。これを、**関数のプロトタイプ宣言**といいます。

ここでは、関数のプロトタイプ宣言を使ったソースコードを紹介します。

```
/* MyC16.c */
#include <stdio.h>

int Tashizan(int, int);

int Tashizan(int a, int b)
{
  int c;

  c = a + b;

  return c;
```

```
13:}
14:
15:int main(void)
16:{
17:  int a;
18:  int b;
19:  int c;
20:
21:  a = 1;
22:  b = 1;
23:  c = Tashizan(a, b);
24:  printf("%d + %d = %d¥n", a, b, c);
25:
26:  a = 2;
27:  b = 2;
28:  c = Tashizan(a, b);
29:  printf("%d + %d = %d¥n", a, b, c);
30:
31:  return 0;
32:}
```

4行目が、関数のプロトタイプ宣言です。プロトタイプ宣言とは、戻り値の型や引数の型などを、4行目のように書くことです。プロトタイプ宣言は、その関数を使う（23行目より）前に書いておかなければなりません。

実は、このプログラムのように、関数を使う（23行目より）前に（6行目から）関数が作ってある場合は、プロトタイプ宣言は必要ありません。つまり、関数のプロトタイプ宣言は、関数の作成とその使い方の順序を考えてやれば必要ないのです。

それでは、なぜ、関数のプロトタイプ宣言が必要になっているのかを説明すると、大きなプログラムをたくさんの人で作るときに、関数の作成とその使い方の順序を細かく考えていられない（考えられない）場合があるからです。

C言語では、関数の作成とその使い方の順序を考えてやるか、そうでない場合は、関数のプロトタイプ宣言が必要だと説明しましたが、これはC言語のコンパイラが、ソースファイルの先頭から最後までを一度しか読まないような設計で作られているからです。

なお、「stdio.h」などのヘッダファイルには、このような関数のプロトタイプ宣言が書かれています。例えば、「stdio.h」には、標準的な入出力関数のプロトタイプ宣言が書かれています。

標準関数

標準関数とは、C言語に最初から用意されている関数のことです。C言語が用意してくれている点が違うだけで、いままで作成してきた関数と変わるところはありません。

標準関数は、C言語でプログラムを作るときにあらかじめ用意しておくと便利な関数の集まりです。その種類は、次のものがあります。

関数の種類	入出力関数
	数学関数
	文字列操作関数

これらはどんなC言語でも、必ず同じ機能を提供する関数が用意されています。つまり、標準関数を使うことにより、異なる開発環境でもソースコードを修正することなく、コンパイルしてプログラムを実行できるのです。ちなみに、このような言語の特徴を、**移植性がよい**といいます。

標準関数を使うには、その標準関数のプロトタイプ宣言が書いてあるヘッダファイルを、ソースファイルの先頭でインクルードします。

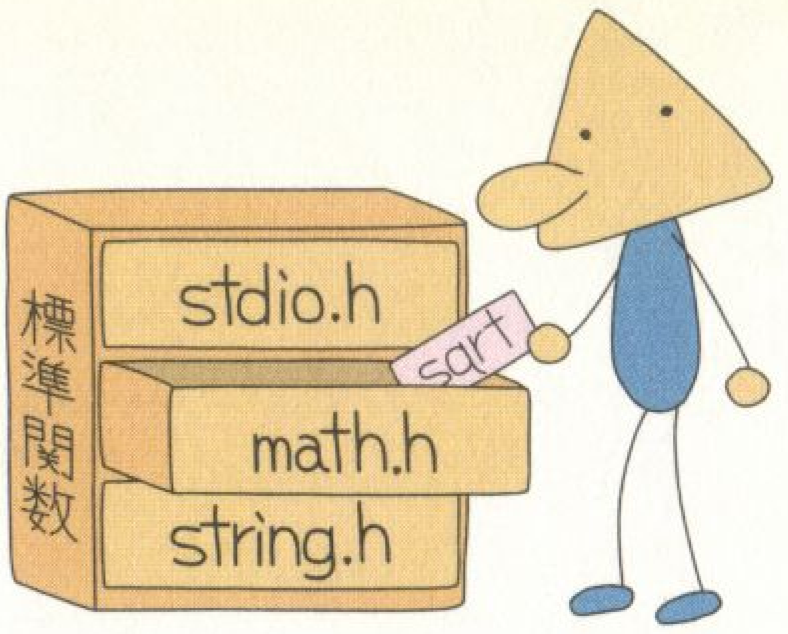
```
#include <ヘッダファイル名>
```
書式

例えば、「printf」では「stdio.h」をインクルードしましたね。

```
#include <stdio.h>
```
書式

上に書いた、標準関数のプロトタイプ宣言が書いてあるヘッダファイルは、次のようになります。

関数の種類	
入出力関数	stdio.h
数学関数	math.h
文字列操作関数	string.h



ここでは、数学関数を使って、平方根を計算するソースコードを紹介します。

```
/* MyC17.c */
#include <stdio.h>
#include <math.h>

int main(void)
{
  double a;
  double b;

  a = 2.0;
  b = sqrt(a);
  printf("aの平方根は%fです¥n", b);

  return 0;
}
```

→ 平方根を計算する関数「sqrt」を使うために、ヘッダファイル「math.h」をインクルード

なお、同じヘッダファイルをインクルードする関数が複数あるときも、ヘッダファイルのインクルードは一度だけでかまいません。

Section 02
ポインタ

「ポインタ」は、C言語をはじめて学ぶときに、最も理解しにくい考え方ですが、その使い方をマスターすると、C言語らしいプログラムが書けるようになります。ここでは、「ポインタ」について、はじめの一歩がわかるようにしましょう。

ポインタ

C言語では、アドレスを記憶するだけの変数を作ることも可能です。この変数を**ポインタ**といいます。変数のアドレス（記憶している場所）を記憶している変数のこととも言います。つまり、変数を「指し示す変数」という意味でポインタといわれているのです。

ポインタの宣言の方法は、次のように変数名の前に「*」を付けます。

書式

```
int *p;
```

ここで注意したいのは、「int」です。これはポインタが、int型なのではありません。int型の変数を指し示すために使うポインタ「p」を宣言したということになるのです。つまり、「どこかにあるint型の変数のアドレスを覚える「p」を用意しますよ」ということです。

わかりにくいと思うので、実際にソースコードで書いて、実行させてみましょう。

```
/* MyC14.c */
#include <stdio.h>

int main(void)
{
  int a;
  int *p;

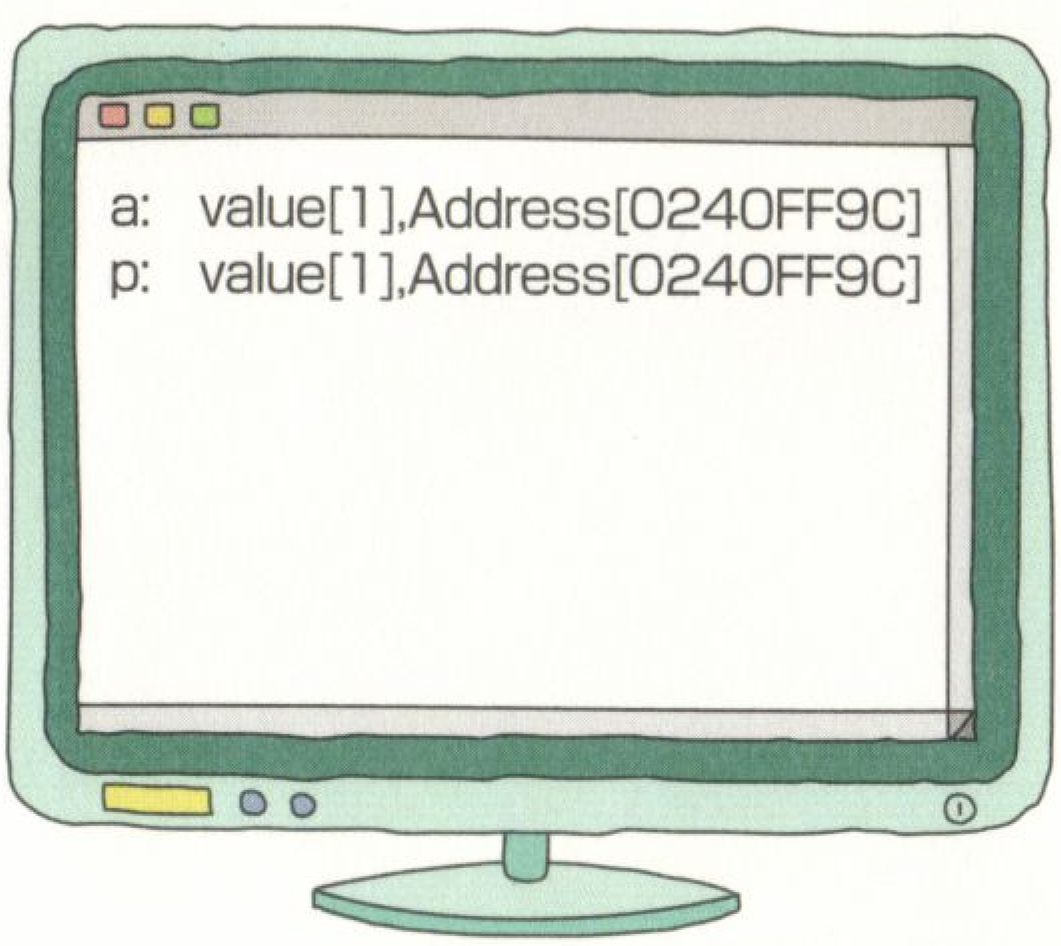
  a = 1;
  p = &a;

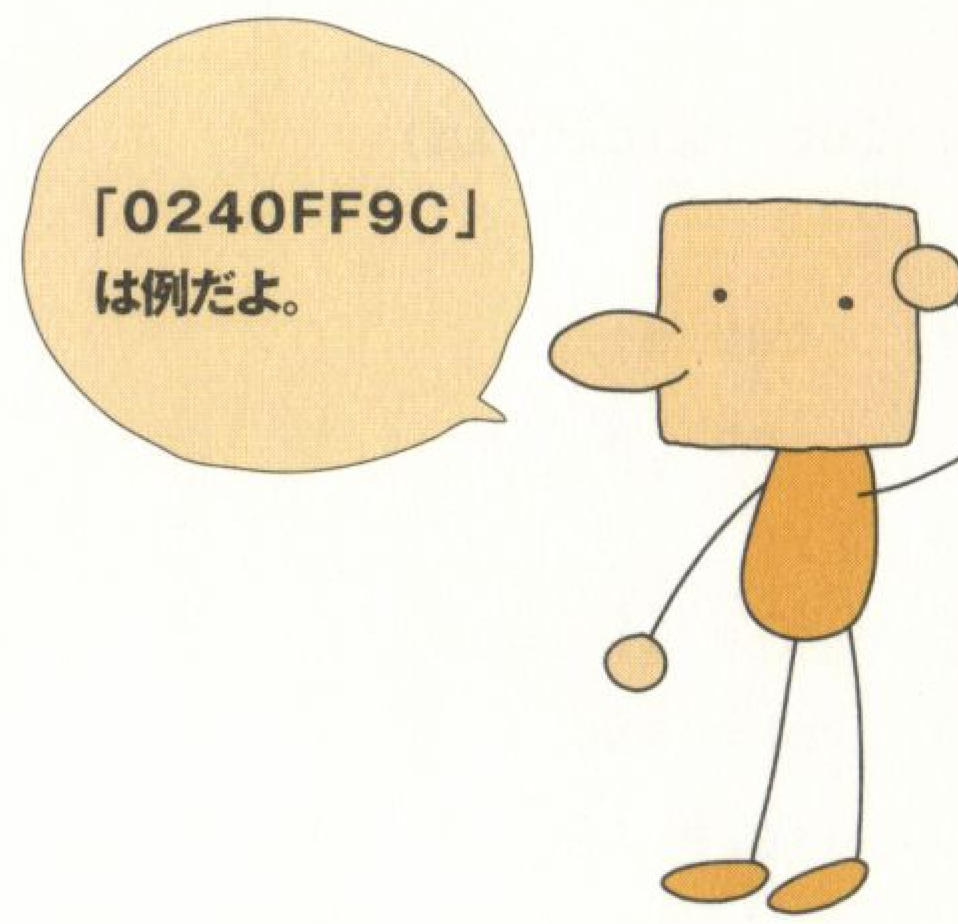
```

```
12:  printf("a: value[%d],Address[%p]¥n", a, &a);
13:  printf("p: value[%d],Address[%p]¥n", *p, p);
14:
15:  return 0;
16:}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC14.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC14.c -o MyC14** Enter」と入力します。
❻「**MyC14** Enter」と入力します。





10行目でポインタ「p」には、変数「a」のアドレスが入ったことになります。このポインタに、13行目にあるように「*」という記号を付けると、変数「a」の値を知ることができます。

▶専門的な説明

13行目の「*」は、**間接演算子**というもので、ポインタを宣言したときに使った「*」と、意味的に区別されます。

ちょっとわかりにくいかもしれませんが、これを順に説明すると、次のようになります。

❶「p = &a;」で、ポインタ「p」に変数「a」のアドレスが入ります。

❷ 関数「printf」中の「*p」は、そのアドレスに割り当てられたint型変数「a」と同じように使うことができます。

この説明を裏付けるのが実行結果です。値とアドレスがまったく同じですね。

例題 ポインタ

次のソースコードを実行すると、どのような結果を画面に表示するか、考えてください。

```
/* MyC15.c */
#include <stdio.h>

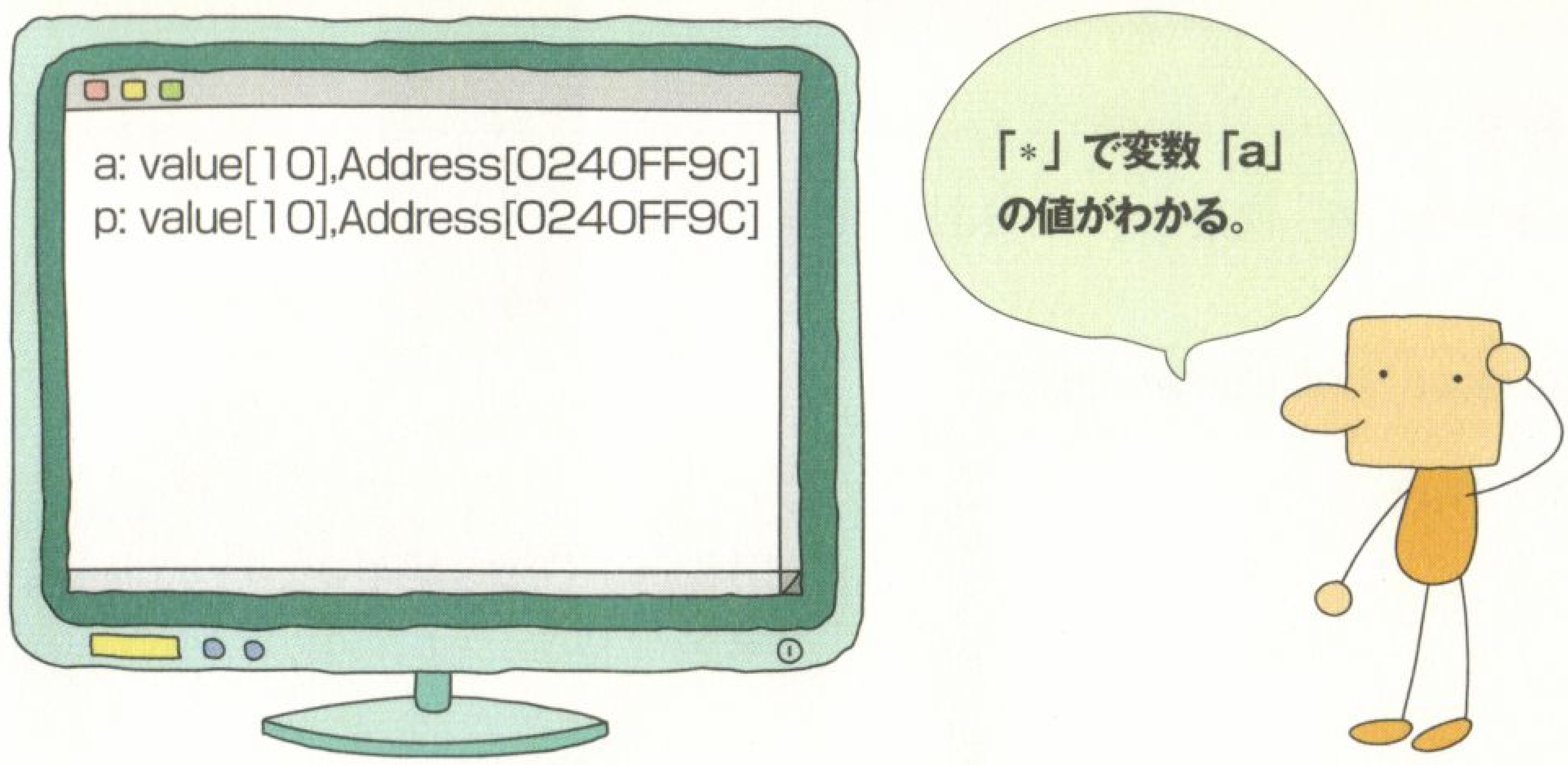
int main(void)
{
  int a;
  int *p;

  a = 1;
  p = &a;
  *p = 10;

  printf("a: value[%d],Address[%p]¥n", a, &a);
  printf("p: value[%d],Address[%p]¥n", *p, p);

  return 0;
}
```

●解答



●解説

10行目を見てください。実は、ポインタ「p」は、「*」を付けると変数「a」の値を知ることができるとともに、「a」そのものになることができるのです。そのため、11行目のようにすると変数「a」の値は、「10」になってしまいます。

ただし、10行目は絶対に省いてはいけません。なぜなら、ポインタ「p」が、メモリ中のどこのアドレスを指しているかわからないからです。もし、このアドレスが、コンピュータに不具合を起こすような場所だった場合、何が起こるかわかりません。最悪の場合、フリーズしてしまう可能性もあります。

このようにポインタ変数を使うときは、10行目のように、必ず変数として確保されたアドレスを指すようにしておくことが絶対条件になります。

▶専門的な説明

Windowsなどの0Sの不具合も、偶然の結果、上のような現象が起こった場合に起こることがほとんどです。

COLUMN 拡張子を表示させる

拡張子は標準の設定では表示されません。拡張子を確認しながら作業を行いたい場合には、エクスプローラーの「フォルダオプション」から拡張子の表示設定を行います。ツールメニューから「フォルダオプション」を選択して、フォルダオプションを表示します。表示タブの詳細設定から「登録されている拡張子は表示しない」のチェックを外します。「すべてのフォルダに適用」ボタンをクリックして設定は完了です。

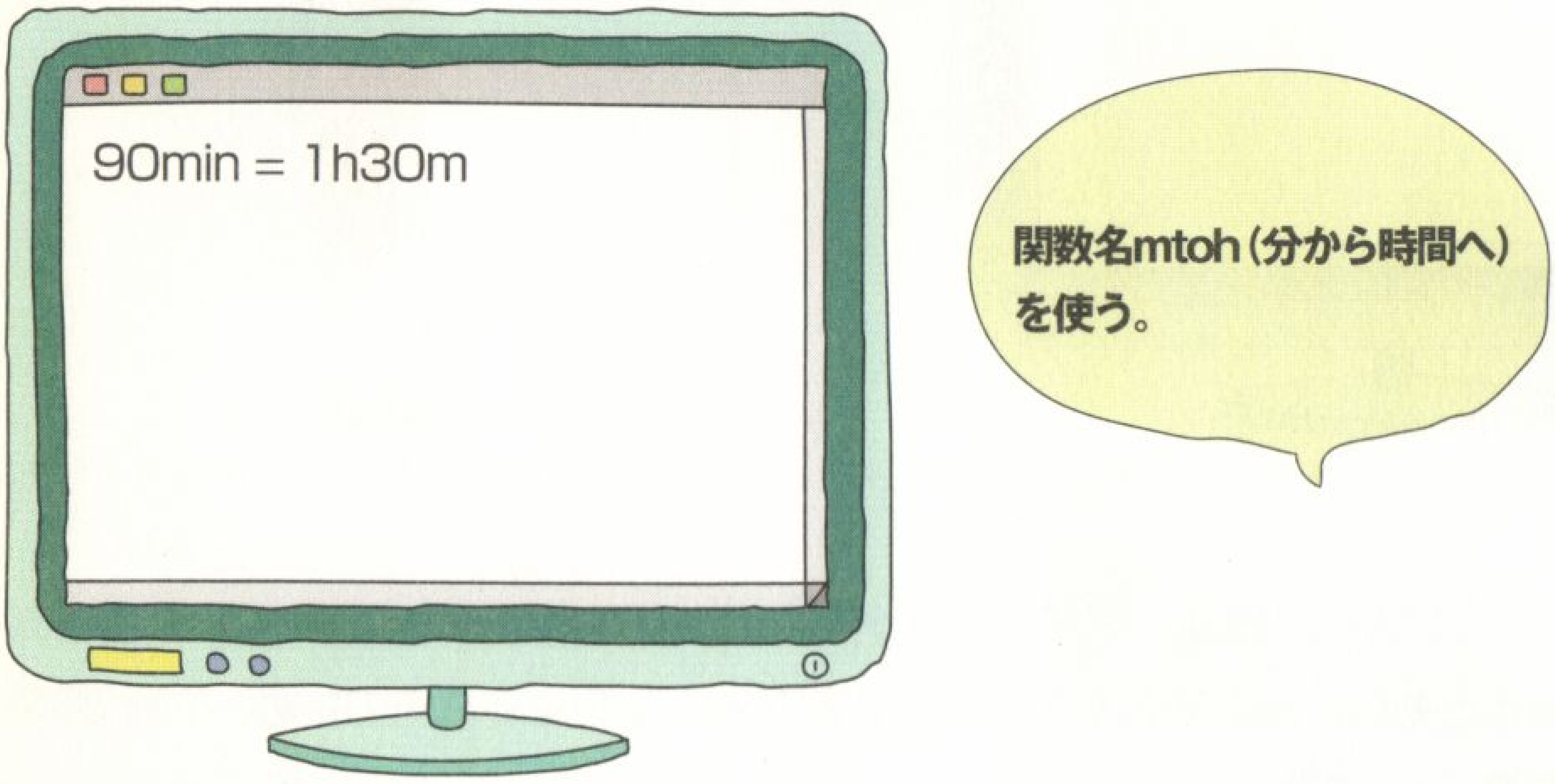
practice 練習問題

解答は213ページ

1 関数を作る　難易度★★★★★

4章の練習問題3を、関数を使って書き直してください。

▶▶ヒント：本文142ページ参照



2 関数を作る　難易度★★★★★

5章の練習問題1を、関数を使って書き直してください。

▶▶ヒント：本文142ページ参照

3 ポインタ　難易度★★★★★

int型の数値を指し示すために使うポインタp「int＊p」に、int型の数値を代入できるでしょうか？

できる場合もできない場合でも、その理由を説明してください。

▶▶ヒント：本文160ページ参照

memo

Chapter6のまとめ

★関数

関数とは、何かをするためにしなければならない一連の処理をまとめて名前を付けたもののことをいいます。

★引数

引数とは、関数に渡すデータのことをいいます。

★戻り値

戻り値とは、関数から受け取るデータのことをいいます。

★標準関数

標準関数とは、C言語に最初から用意されている関数のことをいいます。

★アドレス

アドレスとは、変数の記憶されているメモリ上の位置のことをいいます。

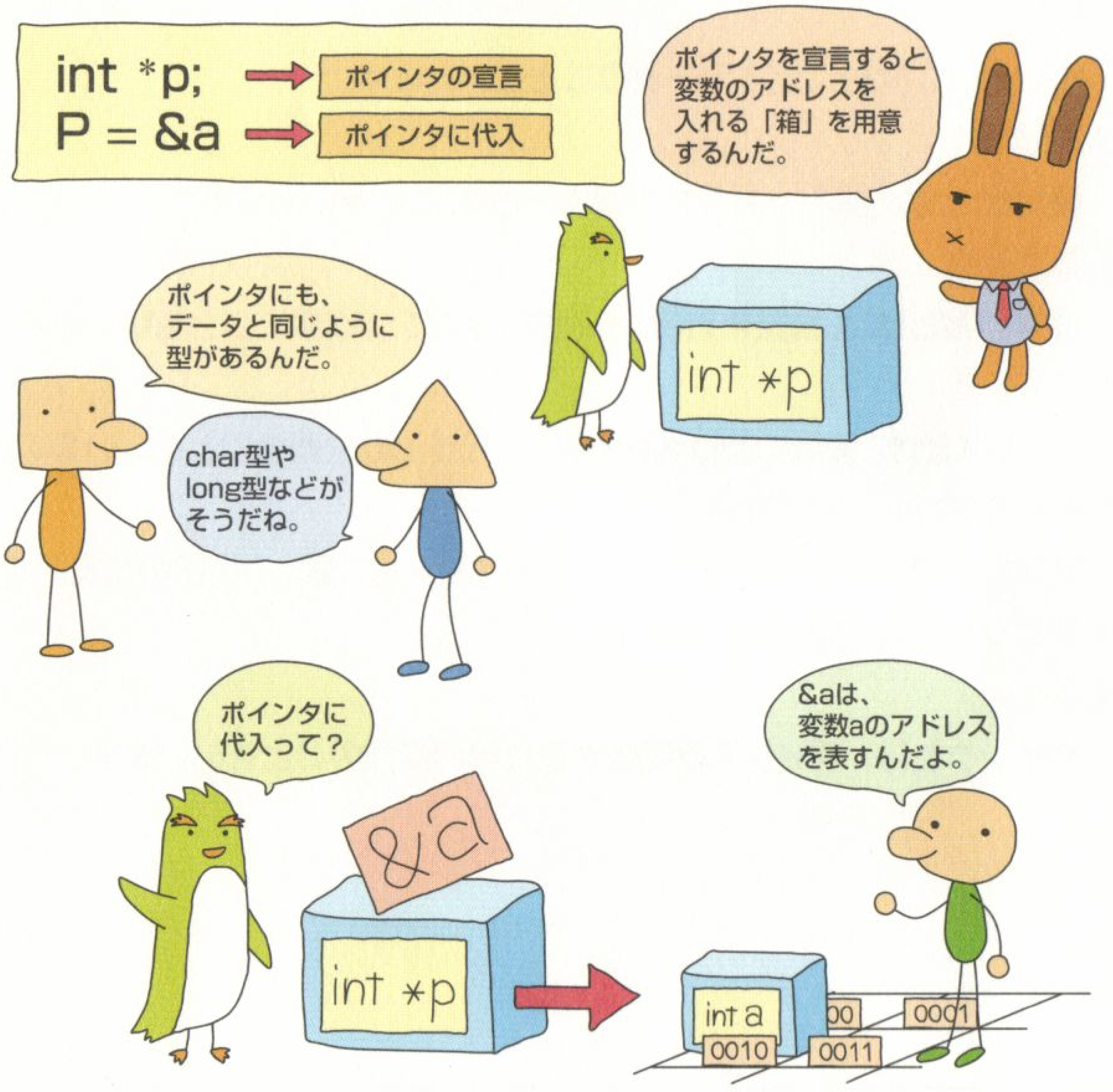
★関数のプロトタイプ宣言

関数のプロトタイプ宣言とは、関数を使う前に関数の引数の型などをチェックする仕組みのことです。

★ポインタ

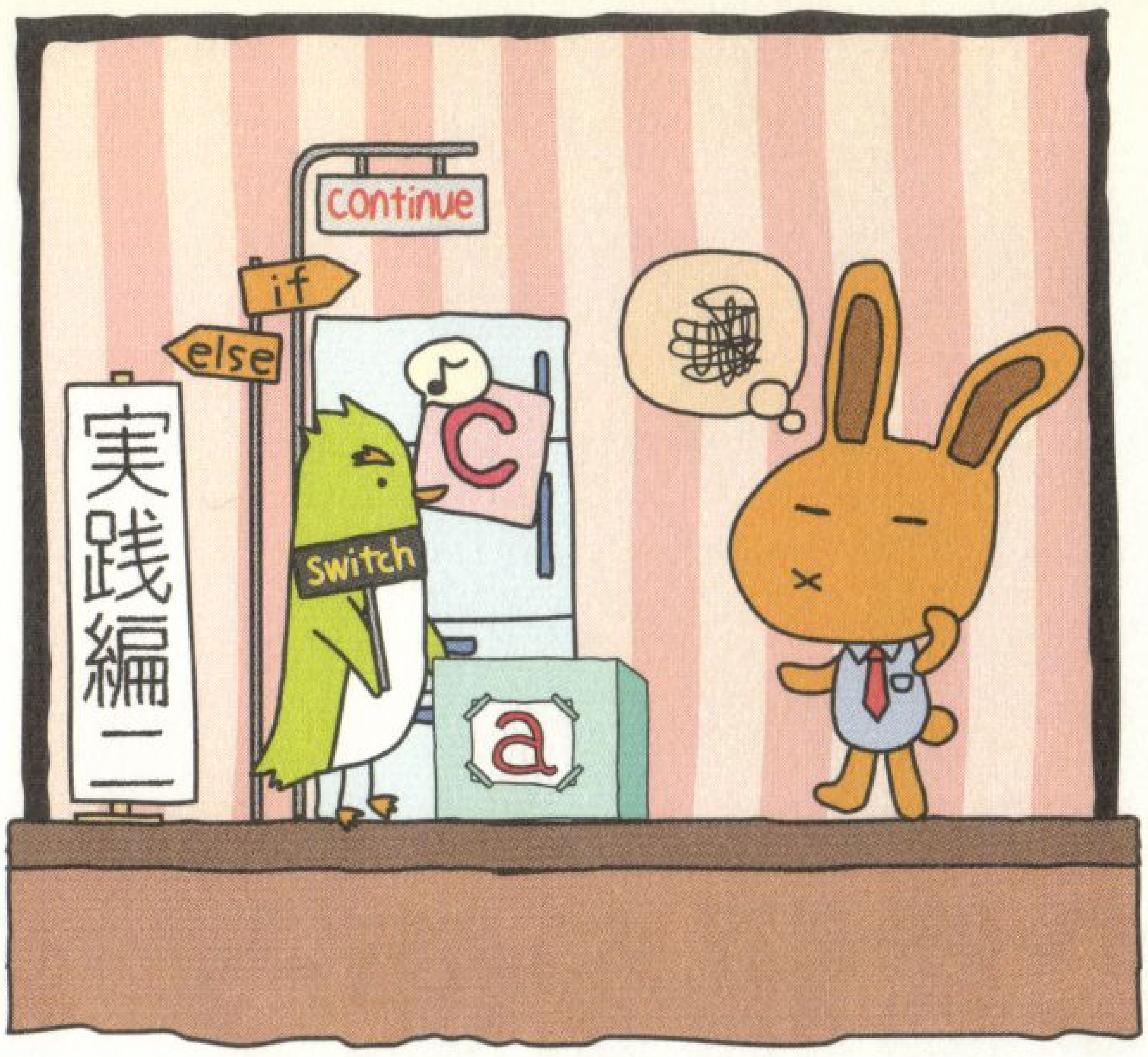
ポインタとは、アドレスを記憶するだけの変数のことをいいます。

COLUMN ポインタを理解しよう！



変数とは、コンピュータが、どれぐらいの大きさのデータをメモリのどこにしまったかをおぼえやすくするために名前を付けたものです。

そして、プログラミングを進めていく上で、コンピュータがデータを、メモリのどこにしまったのかについて知ることが必要になる場合もあります。C言語ではそのメモリの番地を**ポインタ**といいます。



★さあ、もうひとがんばりです。

C言語プログラミングの根本を学び、

C言語の世界へ旅立ちましょう。

★C言語プログラミングの基本を確認したい方は

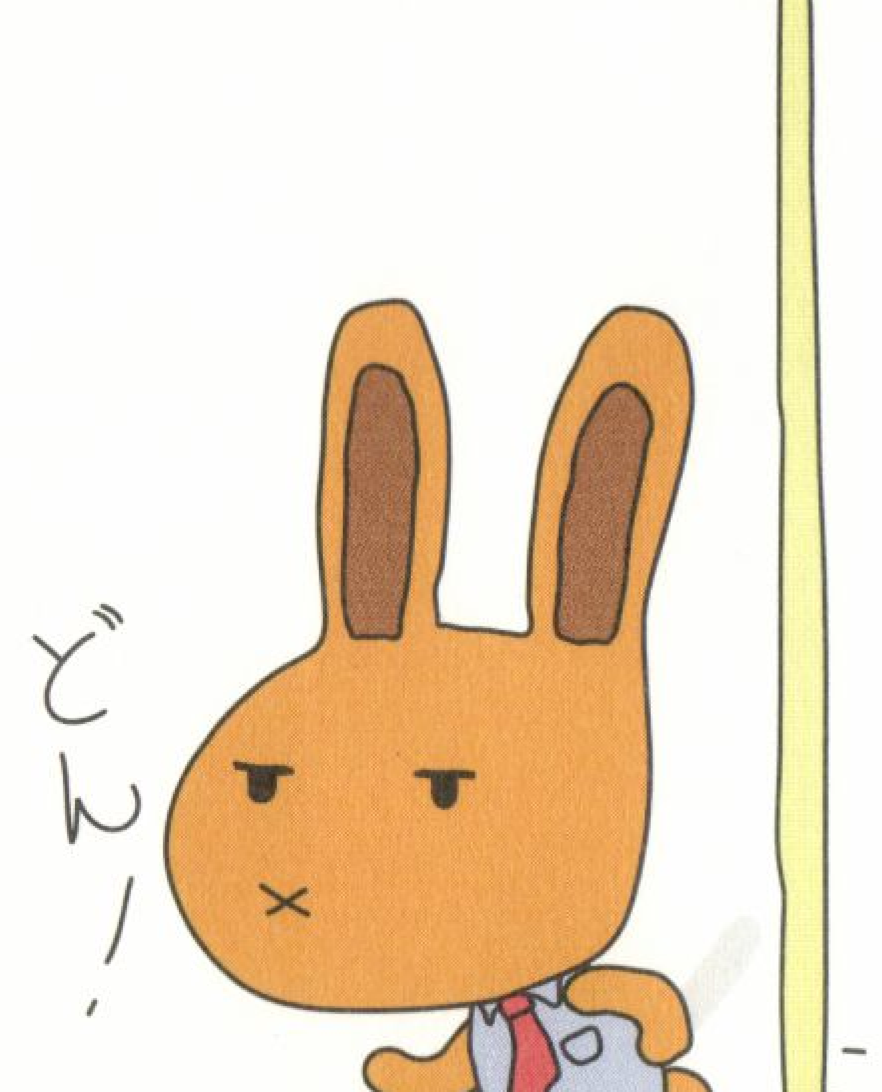
➡ 第3章本文54ページへ

★C言語プログラミングの応用を確認したい方は

➡ 第4章本文72ページへ

Chapter 7

C言語を使う

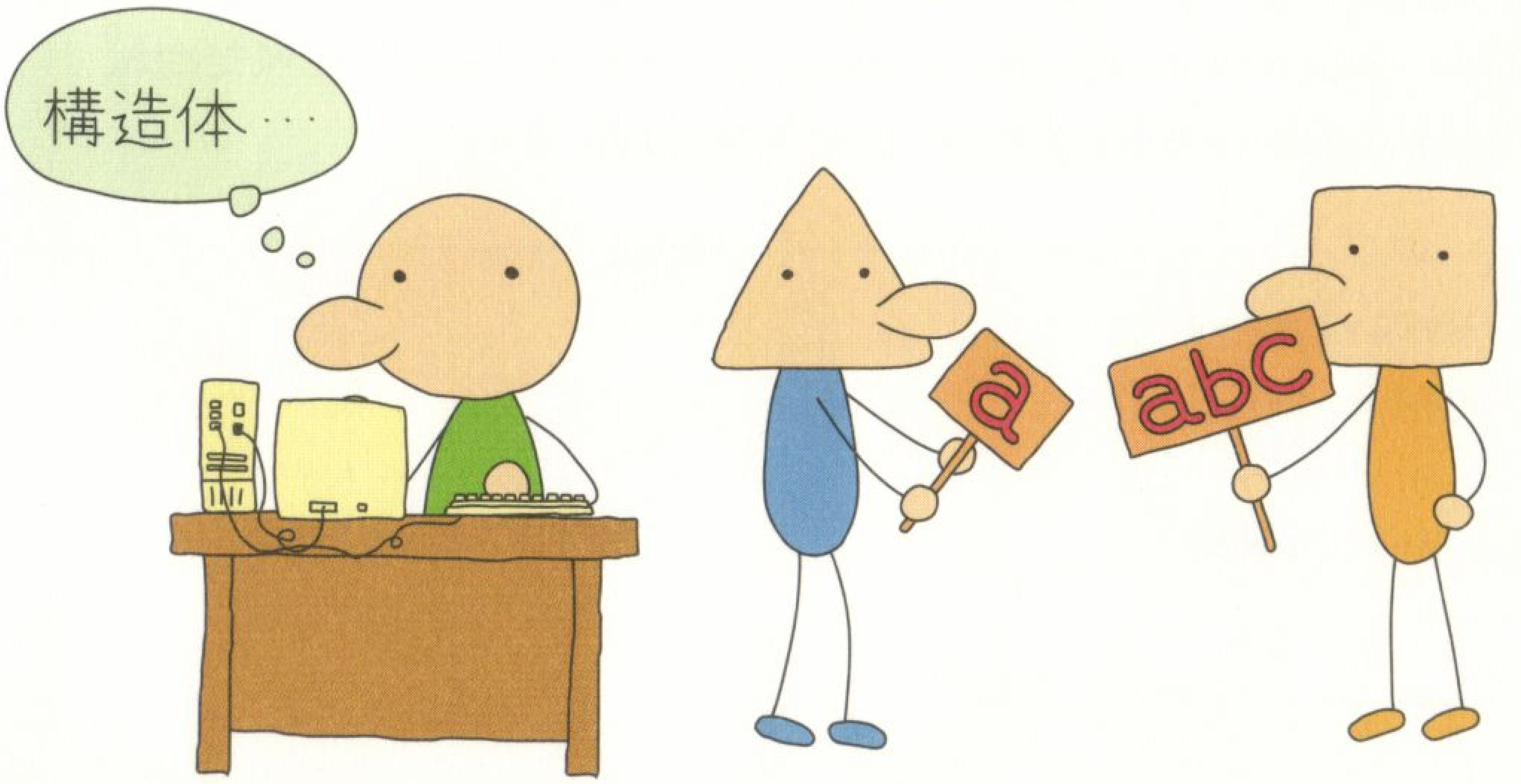


この章でマスターすること

この章からは、本格的にC言語を使っていくことにしましょう。いままでの章は、C言語に限らず、どんなプログラミング言語でも必要な知識を勉強してきたのです。

ここから、やっと本格的なC言語のプログラミングに取りかかれるのです。それでも、ほんの少しだけC言語をのぞいただけに過ぎません。

はじめに文字と文字列について説明します。次に、キーボードからの入力や構造体を学んでいきます。ファイルについておぼえていくうちに、だんだんプログラムらしいプログラムを作ることができるようになるでしょう。



C言語必須用語

文字	文字列
標準入力	標準出力
構造体	ファイル

ステップアップ用語

標準入出力	リンカ
プリプロセッサ	ライブラリ

Section 01 文字と文字列

「文字」とは何でしょうか？　「文字列」とは何でしょうか？　C言語は、これらを違ったものとして扱います。ここでは、「文字」と「文字列」について、説明していきます。

文字列とは

いままで「printf」を使って、当然のことのように画面に文字を表示してきました。ところが、この文字に関わることが、C言語ではかなり重要なことなのです。

まずは簡単なことから説明していきます。

C言語において、**文字**とは、「a」のように1文字の字のことをいいます。また**文字列**とは、1文字以上の文字がつながったもののことをいいます。

例えば、「abcde」は文字列ですが、「a」も1文字以上なので文字列になります。ちょっとわかりにくいかもしれませんが、少しづつ説明していきますのでお付き合いください。

それでは、まず「文字」を表示するソースコードから説明します。

```
/* MyC181.c */
#include <stdio.h>

int main(void)
{
  char c;          → 文字型の変数「c」を宣言

  c = 'a';         → 文字型の変数「c」に、文字「a」を代入
  printf("%c", c); → 文字型の変数「c」の値を画面に表示

  return 0;
}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。

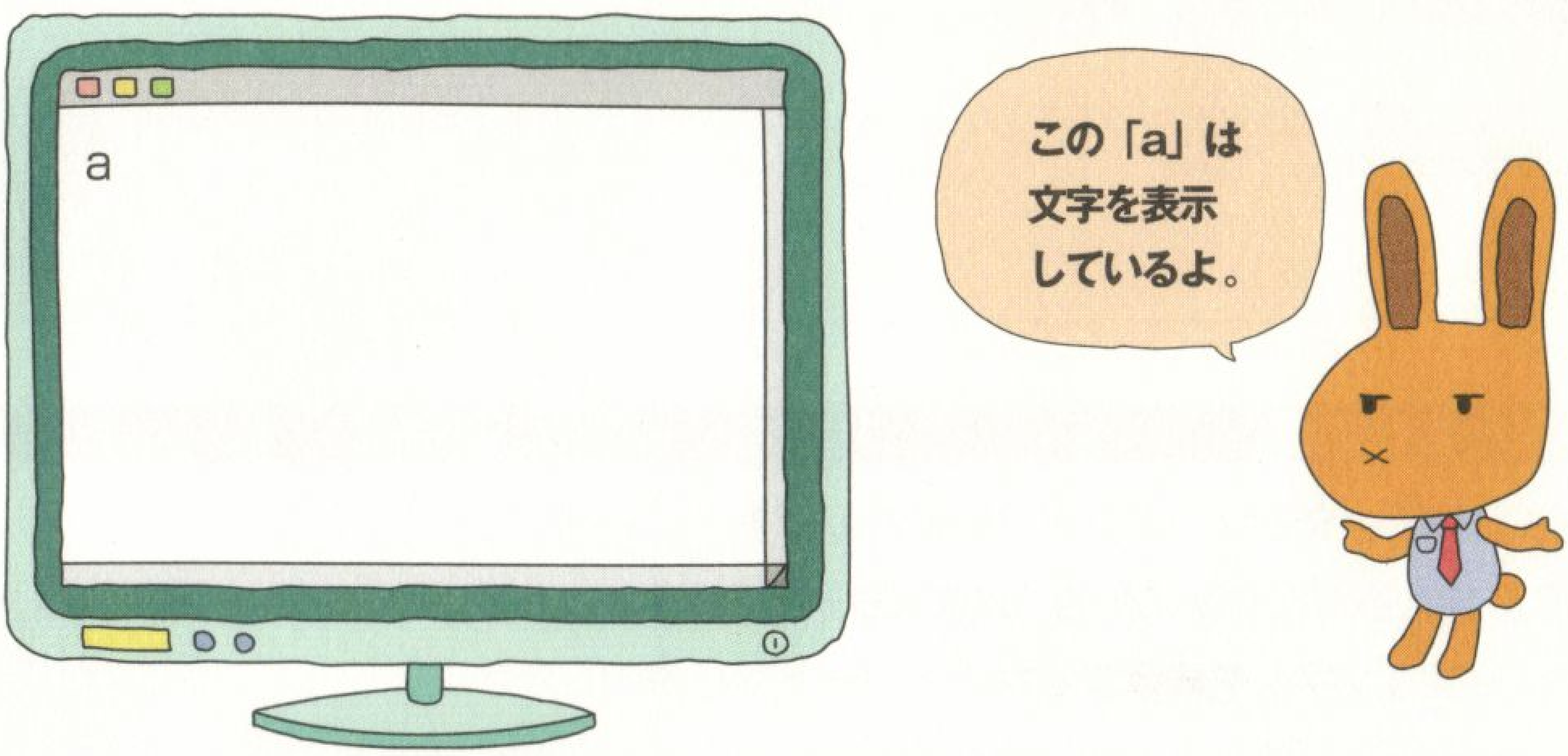
❷「MyC」フォルダにファイル名「MyC181.c」で保存します。

❸ コマンドプロンプトを起動します。

❹「**cd c:¥MyC** Enter」と入力してください。

❺「**gcc MyC181.c -o MyC181** Enter」と入力します。

❻「**MyC181** Enter」と入力します。



6行目で、文字型の変数「c」（1バイト）を宣言しています。

8行目では、変数「c」に文字「a」を代入しています。このとき、変数「c」には、文字の文字コードが代入されます。ソースコード中で「文字」を表すには、「'（**シングルクォーテーション**）」で囲む必要があります。ちなみに、「"（**ダブルクォーテーション**）」で囲まれたのは「文字列」です。

9行目で、変数「c」の値を画面に表示しています。文字を表示するには、「printf」に「%c」と指定します。

これで、画面に「文字」を表示させることができるようになりました。また、「文字」を文字型の変数に代入することも理解できたと思います。

次に、「文字列」について考えてみましょう。「文字列」というのは、文字のつながったもののことです。例えば、「abc」という文字の列のことです。

実は、「文字列」は配列で表します。「abc」という文字列を表したいときは、「文字」を扱う文字型の配列を作ればよいのです。

それでは、説明の前にソースコードから見ていきましょう。

```c
/* MyC182.c */
#include <stdio.h>

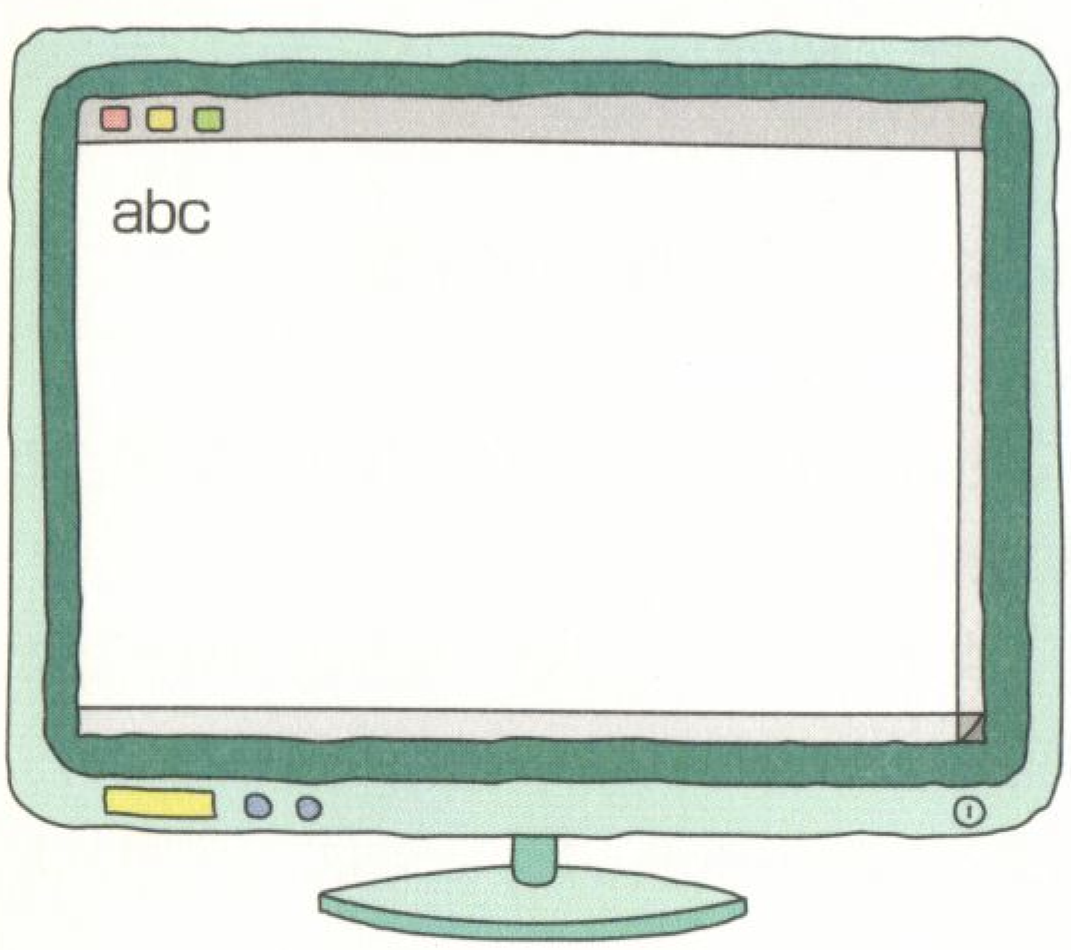
int main(void)
{
  char s[4] = "abc";

  printf("%s¥n", s);

  return 0;
}
```

▶作成手順と実行結果

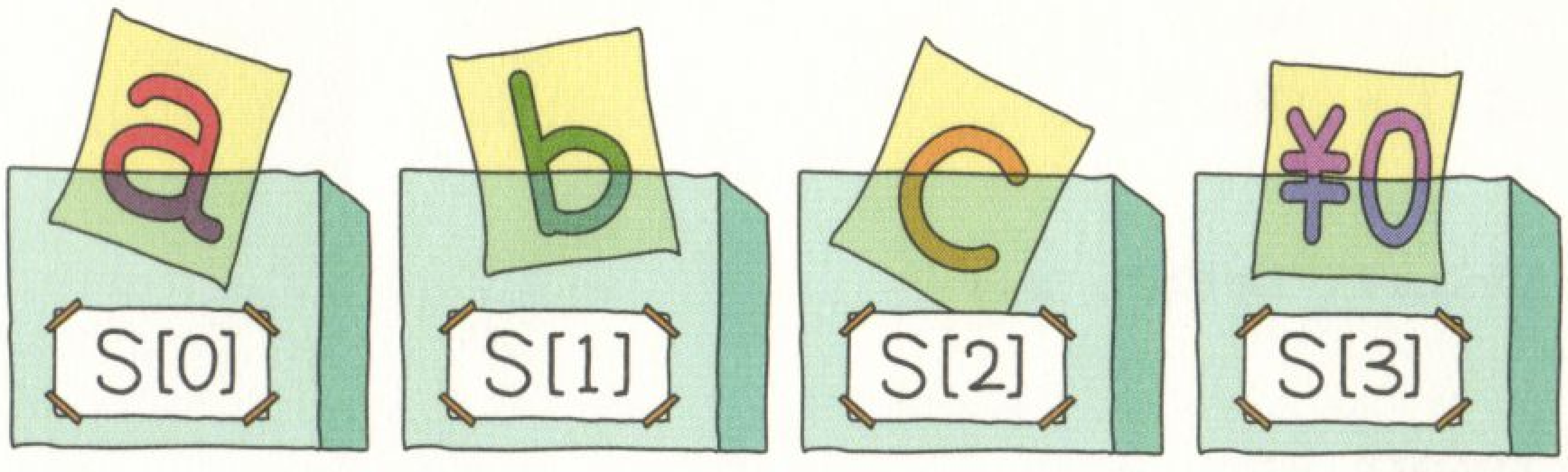
❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC181.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC182.c -o MyC182** Enter」と入力します。
❻「**MyC182** Enter」と入力します。





6行目では、宣言のときに「"abc"」を代入しているように見えますが、前に書いたとおり、これは「初期化」です。

文字型の配列「s」を「"abc"」で初期化すると、次のように、先頭から「'a'」、「'b'」、「'c'」、「'¥0'」が書かれます。



ところで「'¥0'」とは、何でしょうか？



これは「**ヌル**（何もない）**文字**」というもので、実際には「0」という数値です。「ヌル文字」はC言語では、文字列を扱うときに非常に重要で、文字列の終わりを判断するために使われます。そのために配列sでは、「abc」とこの「ヌル文字」も含めた、4つぶんの領域を確保するようにしていたわけです。

文字列「"a"」と文字「'a'」の違いは、ヌル文字があるかないかということです。つまり、見た目は、同じ1文字の「a」ですが、「"a"」にはヌル文字がくっついているので、文字列だったのです。

これで最初に話した、「文字」と「文字列」の違いが理解できたと思います。

8行目では、「printf」に「%s」を指定しています。こうすると、文字列を表示することができます。このとき、後ろには配列の添え字は書かずに配列名だけを書けばよいので、「s」とだけ書いてあります。

さて、ソースコード「My182C.c」ですが、次のソースコードと、内容と実行結果はまったく変わりません。宣言文と初期化を分けて書いてみました。

```
/* MyC183.c */
#include <stdio.h>

int main(void)
{
```

```
    char s[4];

    s[0] = 'a';
    s[1] = 'b';
    s[2] = 'c';
    s[3] = '¥0';

    printf("%s¥n", s);

    return 0;
}
```



このソースコード「MyC183.c」を見れば、文字と文字列の違いがよくわかりますね。

▶専門的な説明

さて、ここで話してきました「文字」とは、C言語では「英数記号のみ」についてです。日本語の文字は、例えば、1文字であっても常に「文字列」となります。

例題 文字と文字列

文字と文字列の違いについて説明してください。

●解答

1文字とは、1バイトの文字コードをいいます。例えば、「'A'」のように、シングルクォーテーションで囲んだものです。文字列とは、1文字以上の文字の集まりをいいます。例えば、「"ABC"」のように、ダブルクォーテーションで囲んだものです。

"A"という文字列と'A'という文字は同じように見えますが、これは別のものです。

Section 02

プログラムらしいプログラム

ここでは、より実践的なプログラム（プログラムらしいプログラム）を学んでいきます。さらに、「構造体」というデータを扱う仕組みについて説明していきます。

キーボードからの入力

いままで作ってきたプログラムでは、データの値をソースコードの中に、直接書いていました。これでは、値を変えるたびにプログラムを編集して、コンパイルし直さなければなりません。

そこで、プログラムが動いているときに、キーボードから値を入力できると、扱えるデータの幅が広くなり、プログラムもよりプログラムらしくなるというものです。

▶専門的な説明

キーボードのことを、**標準入力**というのですが、これと同じように**標準出力**、**標準エラー出力**という言葉もあります。また、標準入力と標準出力を合わせて**標準入出力**といいます。

この標準という意味は、「ふつうに考えれば」とでもしておきましょう。ふつうに考えれば、人がコンピュータに何かを入力するときには、キーボードを使いますね（タブレットもあるのですが、これはふつうではないのでしょう）。

そこで、標準入出力とは、プログラムが標準に使う入出力のことで、次のように考えてください。

標準入力	キーボード
標準出力	コマンドラインの画面
標準エラー出力	コマンドラインの画面

キーボードからデータを入力するようにするには、次のようにします。

```
/* MyC191.c */
#include <stdio.h>

int main(void)
{
  char s[10];

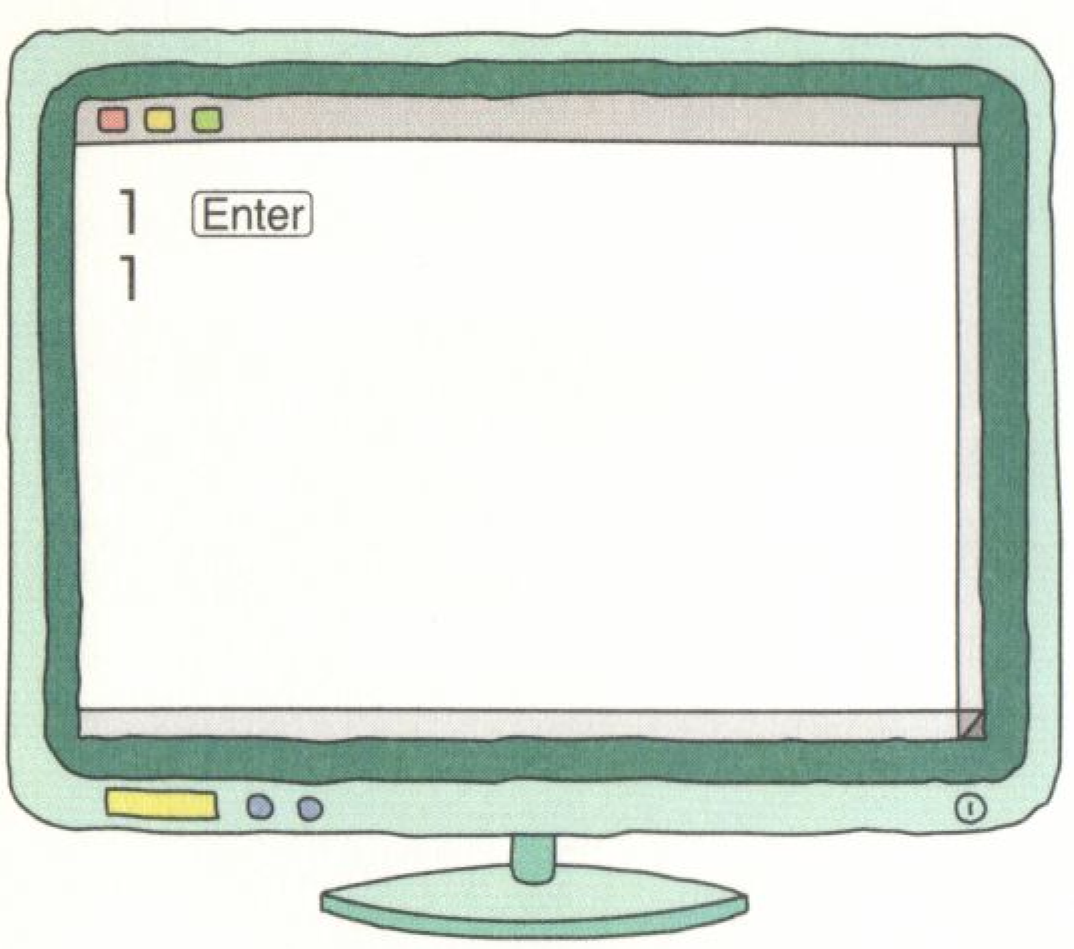
  scanf("%s", s);
  printf("%s¥n", s);

  return 0;
}
```

scanf("%s", s); → 配列「s」に、文字列を記憶する

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC191.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC191.c -o MyC191** Enter」と入力します。
❻「**MyC191** Enter」と入力します。





6行目で、文字型の配列「s」を宣言しています。

8行目が、キーボードからの値を受け取っているところです。「scanf」は、キーボード（標準入力）から入力された、任意の数値や文字を変数に保存します。変換指定子の「%s」は、「printf」と同じで、文字列を表しています。

「printf」の場合は、文字列を表示するという意味でしたが、「scanf」では、文字列が入力されるという意味を示しています。そして、入力された文字列は、文字型で宣言された配列「s」に格納されます。

このプログラムでは、読み込まれるものは文字列でした。つまり、キーボードから「1」と入力した場合でも、それは整数の「1」ではなくて、文字列の「1」として扱われるのです。

これを整数として扱いたい場合は、次のようにします。

```
/* MyC192.c */
#include <stdio.h>

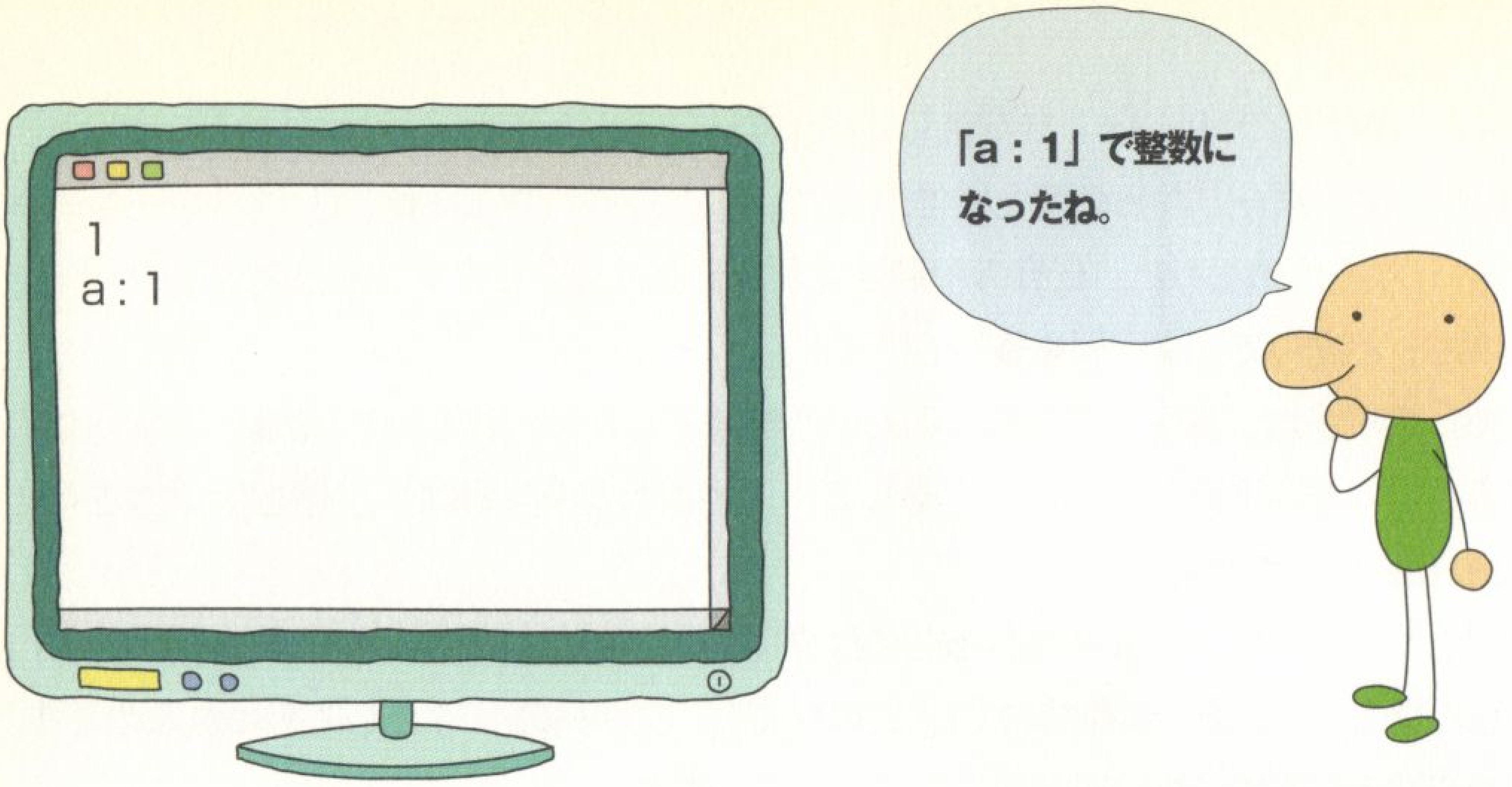
int main(void)
{
  int a;

  scanf("%d", &a);
  printf("a : %d¥n", a);

  return 0;
}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC192.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC192.c -o MyC192** Enter」と入力します。
❻「**MyC192** Enter」と入力します。



ここからの説明は、ちょっと難しいのですが、がんばってください。

8行目の「scanf」の中に、2つの引数があります。最初の引数は「%d」です。これは「キーボードからの入力を int 型の整数に変換せよ」ということです。次の引数は「&a」です。「&a」は、「本文154ページ アドレス」で説明したように、「a」のアドレスになります（変数「a」は、メモリのどこかに存在しており、アドレスによって管理されています）。

▶専門的な説明

ソースコード「MyC191.c」の8、9行目では、「%s」を使っていました。そして引数には、「s」とだけ書いてあります。これは、「s」が文字列配列の先頭アドレスを持っている、ということだったのです。

「scanf」は、キーボードから入力したデータをint型の整数値に変換して、変数「a」の「アドレス」に格納します。

例題 標準入力

ソースコード「MyC4.c」を、標準入力を使って改良してみましょう。

●解答例

```
/* Exercise16.c */
#include <stdio.h>

```

```c
int main(void)
{
  int a;

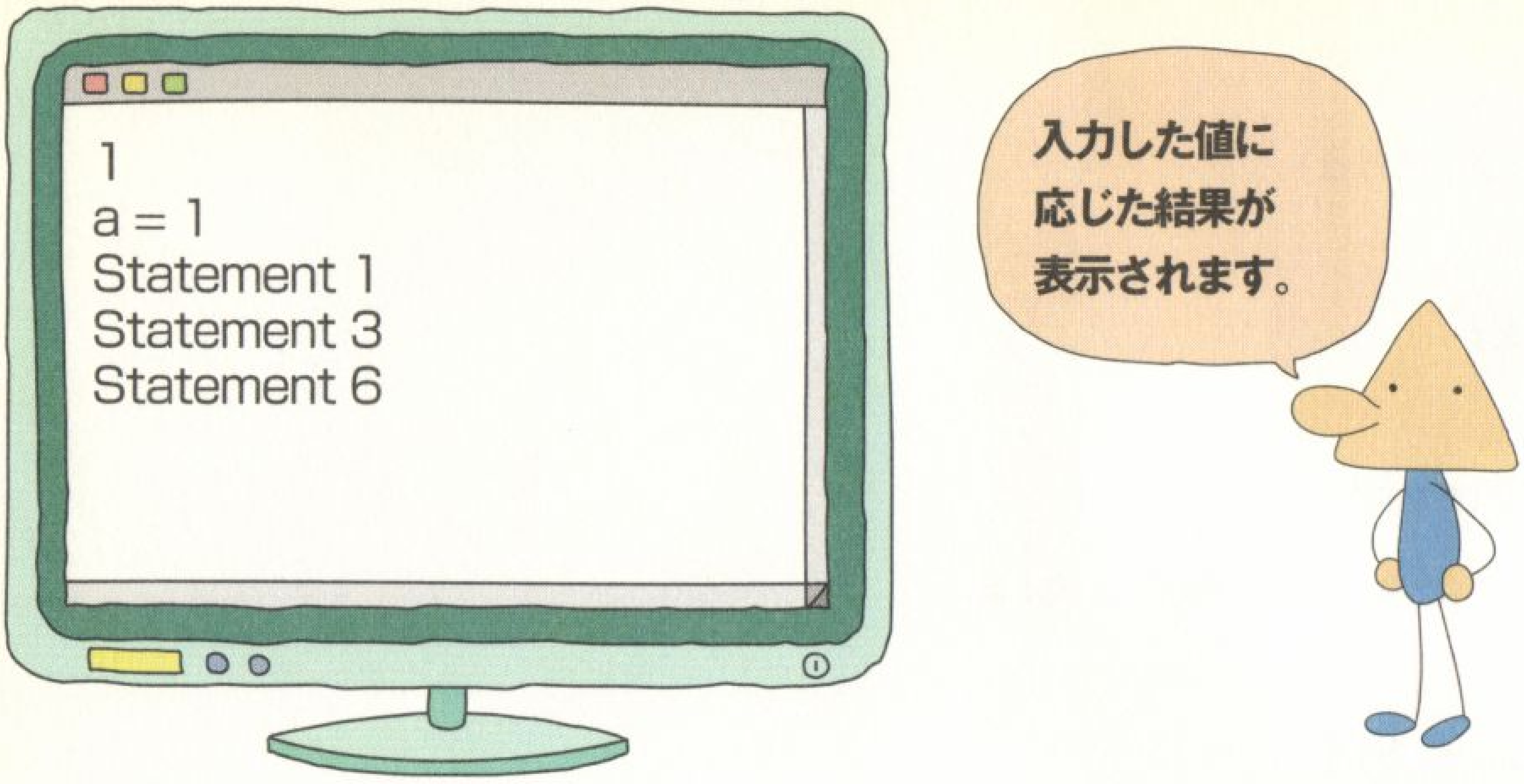
  scanf("%d", &a);
  printf("a = %d¥n", a);

  if (a < 5){
        printf("Statement 1¥n");
  }

  if (a > 5){
        printf("Statement 2¥n");
  }else{
        printf("Statement 3¥n");
  }

  if (a > 5){
        printf("Statement 4¥n");
  }else if (a < 0){
        printf("Statement 5¥n");
  }else{
        printf("Statement 6¥n");
  }

  return 0;
}
```



配列だけではものたりない

本文132ページで、多次元配列について説明しました。そのとき、生徒の「体重」と「身長」を表すのに、2次元配列を使いました。2次元配列を使えば、「体重」だけでなく「身長」も表すことができましたが、さらに生徒の「名前」も表示したいというときはどうすればよいでしょうか？

「体重」と「身長」は、どちらも数値で表せるのですが、「名前」となると、数値で表すことはできません。

これは、日常でも目にすることができます。例えば、清涼飲料水のペットボトルの表記ラベルを見てください。「品名」、「原材料」などは数値で表すことはできません。

このように「もの」には、それを表す様々な「項目」があり、数値で表せるものや文字で構成されているものなどがあります。

そこで、コンピュータに効率よく処理してもらうためには、プログラム言語に、こういった「項目」を簡単に扱えるような仕組みが必要です。

まずは、次のソースコードを見てください。



構造体

```
/* MyC201.c */
#include <stdio.h>

```

```
int main(void)
{
  int i;
  int A[5][2];
  char namae[5][10] = {
           "A0",
           "A1",
           "A2",
           "A3",
           "A4"
  };

  A[0][0] = 20;
  A[0][1] = 120;
  A[1][0] = 21;
  A[1][1] = 121;
  A[2][0] = 22;
  A[2][1] = 122;
  A[3][0] = 23;
  A[3][1] = 123;
  A[4][0] = 24;
  A[4][1] = 124;

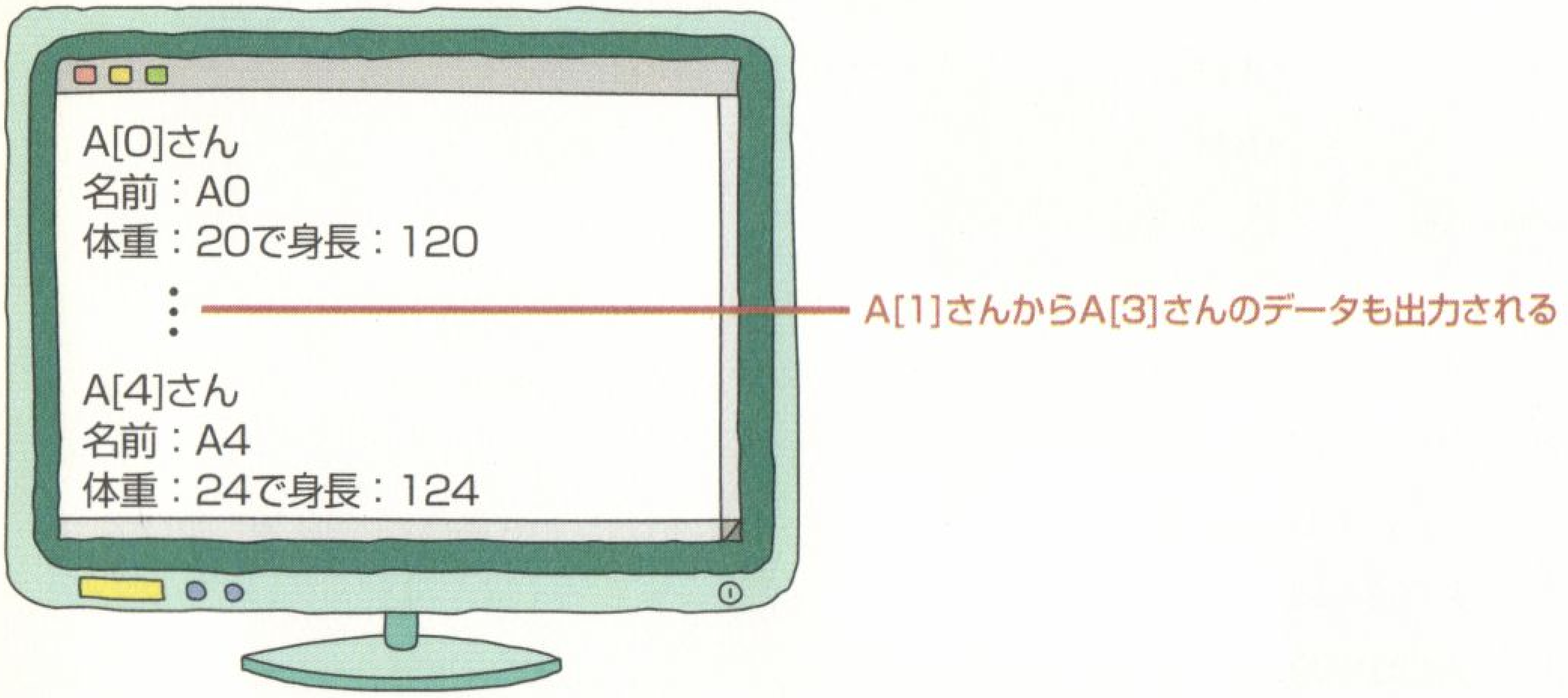
  for (i = 0; i<=4; i++) {
          printf("A[%d]さん¥n", i);
          printf("名前:%s¥n", namae[i]);
          printf("体重:%dで身長:%d¥n", A[i][0], A[i][1]);
  }

  return 0;
}
```

→2次元の文字配列

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC201.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「`cd c:¥MyC` Enter」と入力してください。
❺「`gcc MyC201.c -o MyC201` Enter」と入力します。
❻「`MyC201` Enter」と入力します。



8行目から14行目は、2次元の文字型配列を初期化しています。文字列そのものが配列ですから、文字列の配列は、2次元配列になりますね。値の書き方は、こういうものだとおぼえておきましょう。

ソースコード「MyC201.c」のようにすれば、生徒の名前も表示できますが、表示しなければならない生徒の「項目」が増えていくと、ソースコードの中でどの変数がどの「データ」を表しているかがだんだんわかりにくくなってしまい、そのぶんソースコードの間違い（バグ）も増えていってしまいます。

C言語には、こういった「データ」を簡単に扱えるような仕組みとして、**構造体**というものが用意されています。

「構造体」とは、型の異なるいくつかの変数をまとめたもののことをいいます。つまり、性質の異なるいくつかの変数をまとめてグループ化したもの、と考えることもできます。



まずは、構造体を使った簡単な例から説明します。

では、次のソースコードを書いてみましょう。

```
/* MyC202.c */
#include<stdio.h>

struct MyStruct {      → 構造体の宣言
    int a;
    int b;
};

int main(void)
{
    struct MyStruct s;   → 構造体を使います
    int c;

    s.a = 1;   → 値の代入
    s.b = 2;   → 値の代入

    c = s.a + s.b;

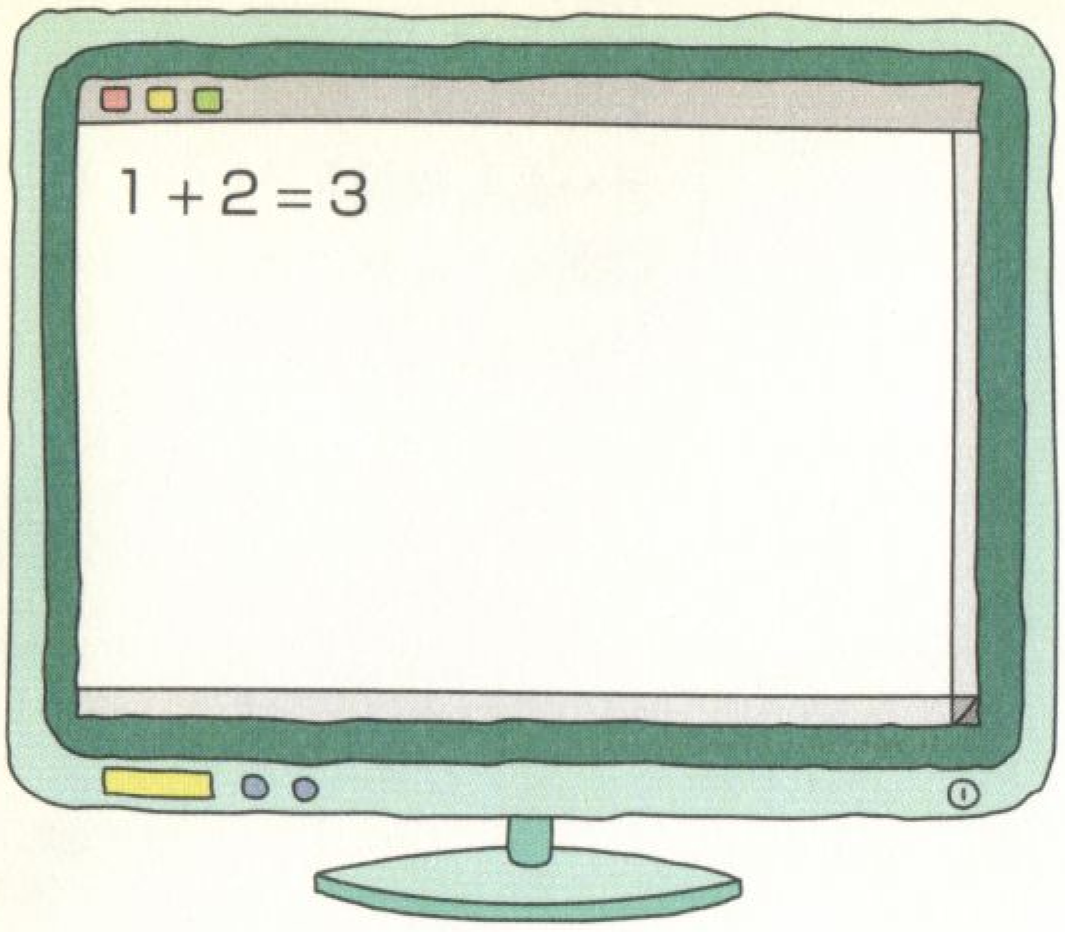
    printf("%d + %d = %d¥n",s.a , s.b , c);

    return 0;
}
```



▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC202.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「`cd c:¥MyC` Enter」と入力してください。
❺「`gcc MyC202.c -o MyC202` Enter」と入力します。
❻「`MyC202` Enter」と入力します。





4行目では、構造体を宣言しています。**構造体の宣言**は、次のように行います。

書式

```
struct 構造体名 {
      変数1;
      変数2;
        ⋮
};
```

「struct」の次の単語が、構造体の名前（**タグ名**といいます）を表し、ブロックの中がまとめられた変数です。

例では、int型を使っていますが、char型や配列変数も使えます。また、この変数のことを**メンバ**といい、「MyStruct」という名前の構造体は、「int型の変数aとint型の変数bをまとめたものです」、ということになります。

注意してほしいのは、「構造体の宣言」は、「変数の宣言」とは違って箱を用意するためのものではありません。「こういうものをこれから作りますよ」、という設計図を書いただけの状態になります。

11行目では、宣言した構造体「MyStruct」を次のように書くことで、実際に構造体を使えるようにしています。

書式

```
struct MyStruct s;
```

これは、構造体「MyStruct」を「s」という名前で使います、ということです。つまり、

「struct MyStruct」という型で、変数「s」を宣言していると考えてもよいでしょう。また「s」は、**構造体変数**といいます。

14、15行目で、値を代入しています。構造体変数とメンバを、「.」でつなぐことでメンバに値を代入することができるようになります。要するに、ここで箱が用意されるのです。

「s.a = 1; 」は、構造体変数「s」の中のメンバ「a」に、「1」を代入するということです。変数を使うときは、次のように書きます。

書式

構造体変数.メンバ名

本文184ページで、メンバに配列を使うことができると説明しましたが、構造体変数にも配列を使うことができます。

```
struct MyStruct s[10];
```

上記のように書くと、これは「s」という構造体を10個用意したことになります。使うときは、次のように書きます。

```
s[1].a
```

これは、2番目の構造体変数のメンバ「a」を表しています。

さて、いよいよソースコード「MyC201.c」を改良していきますが、わかりやすいように2段階で改良していきましょう。

はじめは、まず「構造体の宣言」をして、1人分のデータだけを扱えるようにしてみます。

```
/* MyC203.c */
#include <stdio.h>
#include <string.h>

struct MyStruct {    → 構造体の宣言
    int taijyu;
    int sincho;
    char namae[10];
};

```

```
int main(void)
{
  int i;
  struct MyStruct s;  ——→構造体を使う

  i = 0;

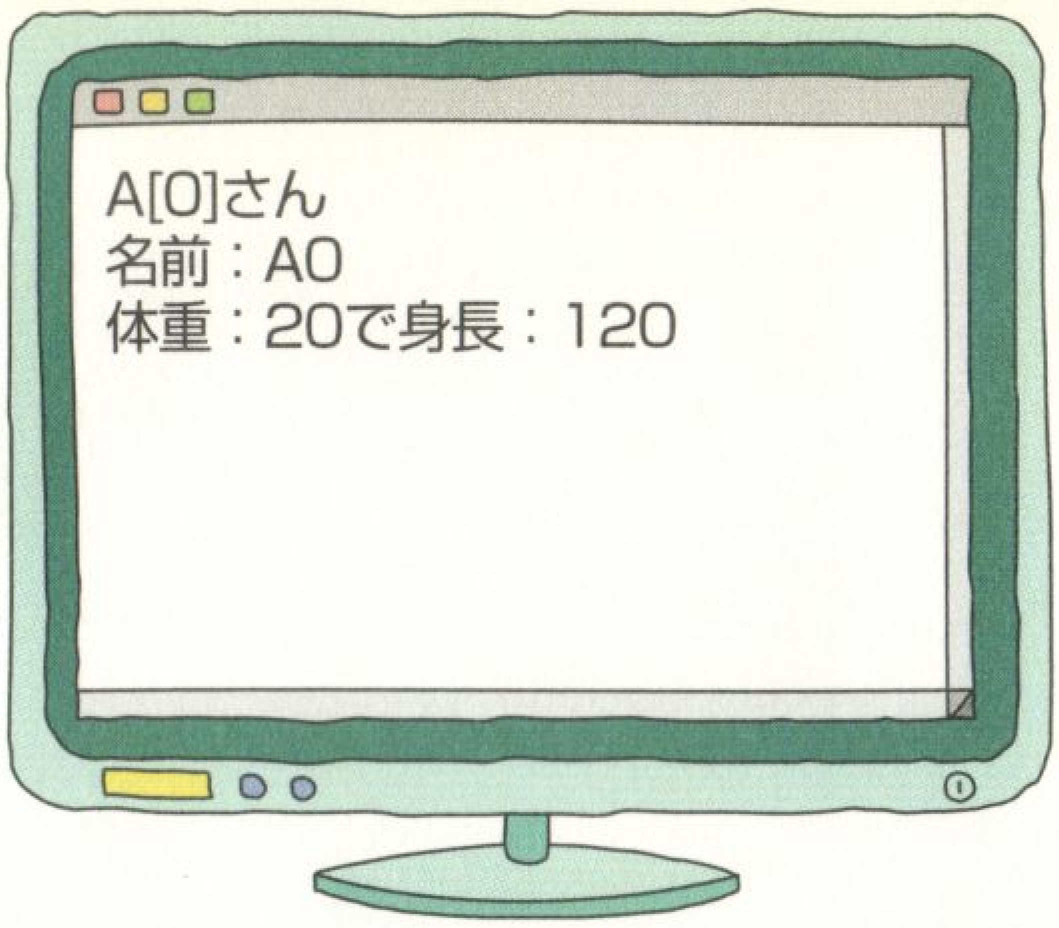
  s.taijyu = 20;
  s.sincho = 120;
  strcpy(s.name, "A0");

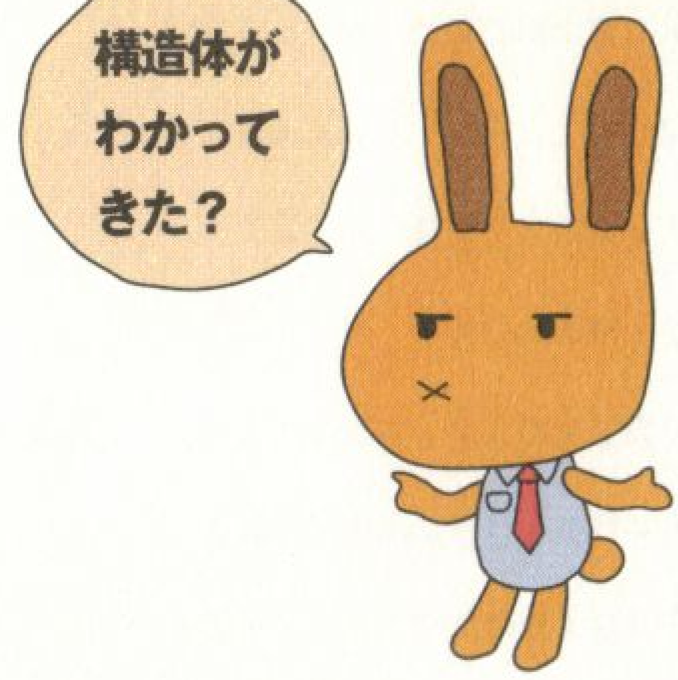
  printf("A[%d]さん¥n", i);
  printf("名前:%s¥n", s.namae);
  printf("体重:%dで身長:%d¥n", s.taijyu, s.sincho);

  return 0;
}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC203.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「`cd c:¥MyC` Enter」と入力してください。
❺「`gcc MyC203.c -o MyC203` Enter」と入力します。
❻「`MyC203` Enter」と入力します。





5行目で、構造体「MyStruct」の宣言をしています。「MyStruct」のメンバには、体重に必要な「taijyu」と、身長に必要な「sincho」、名前に必要な「namae」があります。14行目で、構造体変数「s」を宣言しています。

18行目から20行目で、構造体「MyStruct」のメンバへ値を代入しています。20行目の「strcpy(s.name, "A0");」についてはあとで説明しますので、ここでは、こうやって書くと文字列「A0」が「s.name」に代入されるとおぼえておいてください。

これで、何とか1人分のデータを表示できるようになりましたね。次は、5人分に挑戦します。次のソースコードを書いてみてください。

```
 1:/* MyC204.c */
 2:#include <stdio.h>
 3:#include <string.h>
 4:
 5:struct MyStruct {
 6:    int taijyu;
 7:    int sincho;
 8:    char namae[10];
 9:};
10:
11:int main(void)
12:{
13:  int i;
14:  struct MyStruct s[5];
```

→構造体変数に、配列を使う（5人分）

```
  s[0].taijyu = 20;
  s[0].sincho = 120;
  strcpy(s[0].namae, "A0");
  s[1].taijyu = 21;
  s[1].sincho = 121;
  strcpy(s[1].namae, "A1");
  s[2].taijyu = 22;
  s[2].sincho = 122;
  strcpy(s[2].namae, "A2");
  s[3].taijyu = 23;
  s[3].sincho = 123;
  strcpy(s[3].namae, "A3");
  s[4].taijyu = 24;
  s[4].sincho = 124;
  strcpy(s[4].namae, "A4");

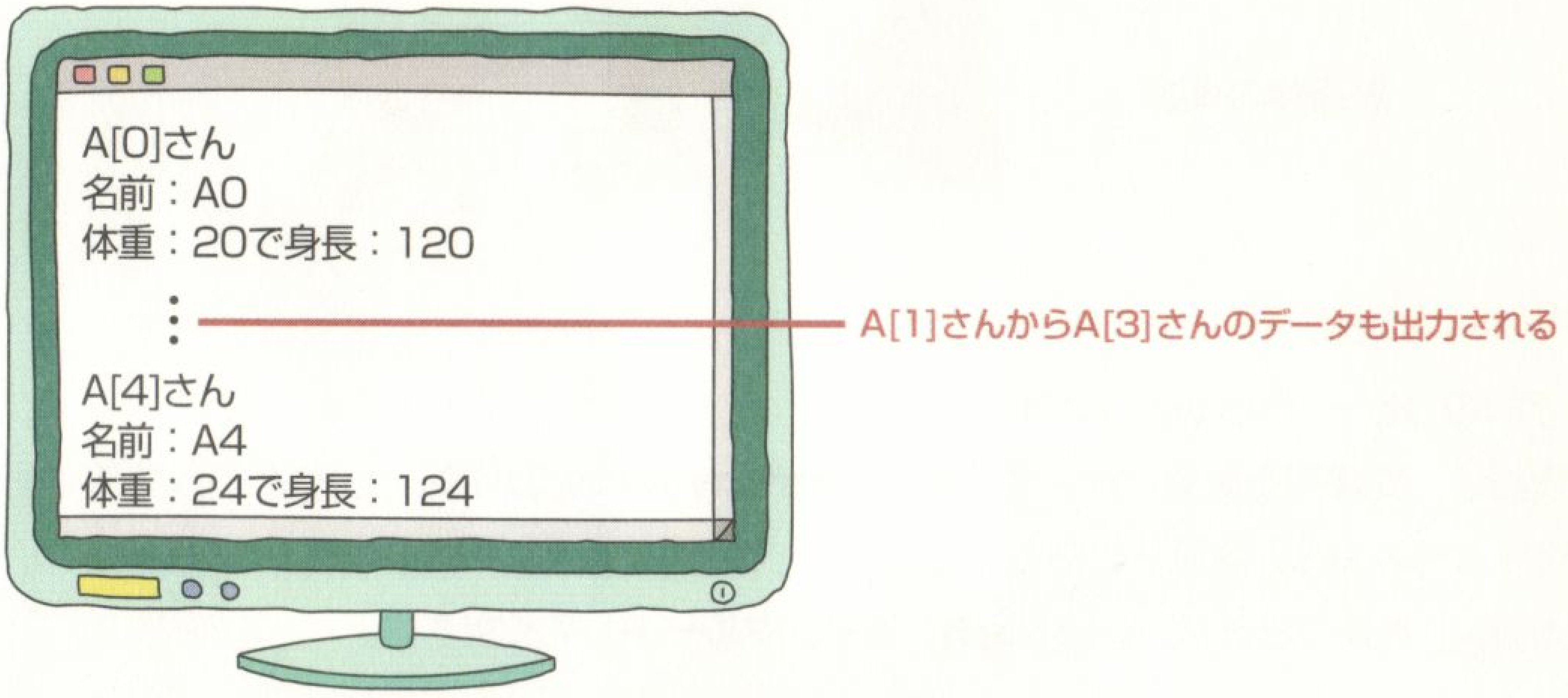
  for (i = 0; i<=4; i++) {
          printf("A[%d]さん¥n", i);
          printf("名前:%s¥n", s[i].namae);
          printf("体重:%dで身長:%d¥n", s[i].taijyu,➡
                                          ➡ s[i].sincho);
  }

  return 0;
}
```

※注意　ソースコード内の➡は、レイアウトの都合により行を折り返していますが、実際には1行で入力してください。

▶作成手順と実行結果

1. テキストエディタでソースコードを入力します。
2. 「MyC」フォルダにファイル名「MyC204.c」で保存します。
3. コマンドプロンプトを起動します。
4. 「**cd c:¥MyC** Enter」と入力してください。
5. 「**gcc MyC204.c -o MyC204** Enter」と入力します。
6. 「**MyC204** Enter」と入力します。

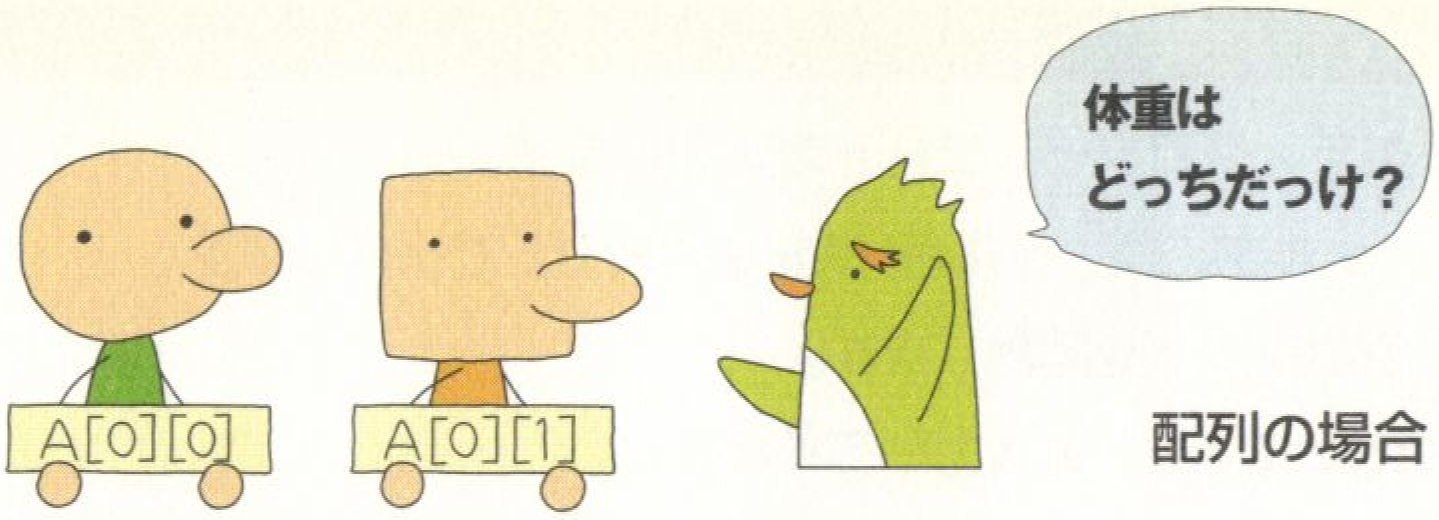


14行目で、構造体変数「s」に配列を使いました。16行目から30行目までは、配列となった構造体変数「s」のメンバに、値を代入してます。

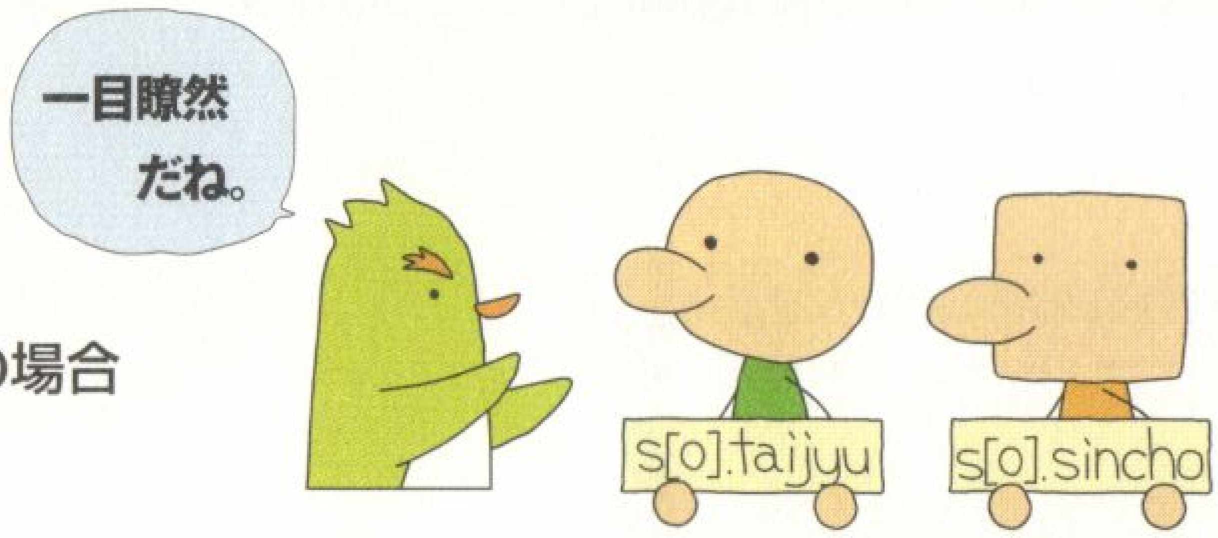
35行目を見てください。ソースコード「MyC111.c」の22行目「printf("体重:%dで身長:%d¥n", A[i][0], A[i][1]);」では、体重と身長を2次元配列として扱っていたので、この行だけを見ると、体重も身長も同じ配列「A」としてしか表せませんでした（体重と身長は、本当は別の項目ですね）。

しかし、ソースコード「MyC204.c」では構造体を使っているので、体重や身長を別の項目として扱うことができるようになり、見た目もわかりやすくなりました。

配列「A」を、「TaijyuToSincho」という名前にすれば何とかなりそうですが、ここでは、「項目や型の違うものを、違うものとして扱う方法がある」ということが重要です。



配列の場合



構造体の場合

▶専門的な説明

●文字列のコピー（strcpy）について

C言語で、文字列を変数に代入するには、**strcpy**(string:文字列 copy)という関数を使います。C言語の代入演算子である「=」には、文字列を代入する機能はありません。

「strcpy」を使うときは、「**string.h**」をインクルードしてください。

例題 構造体

構造体を使って、「住所」、「氏名」、「年齢」、「職業」を表す変数を、ひとまとめにしてみましょう。

●解答例

```
struct MyStruct {
      char Jyusho[20];
      char name[10];
      int nenrei;
      char shokugyo[10];
};
```

ファイルとは

ファイルの操作について説明する前に、まず「ファイル」とは何なのかについて、もう一度確認しておきたいと思います。

ファイルとは、コンピュータによって処理されるデジタル情報としての、文字やデータ、プログラムなどを表します。いままで作成してきたプログラムのソースコードは、まさにファイルといえるでしょう。もっとわかりやすくいえば、デジタル情報ではないのですが、「授業の内容をまとめたノートや手帳のようなもの」だと考えてください。

いままで作成してきたプログラムでは、扱ったデータ（計算結果）を変数に格納していましたが、それだけでは、プログラムが終了すればデータ内容は失われてしまいます。せっかく計算したものが、プログラムの終了とともに消えてしまうのは、もったいないですね。そこで、プログラムが終了してもデータ内容（計算結果）が失われないように、「ファイル」に保存しておけば、見たいときにすぐにデータ内容を見ることができます。

ここでは、ファイルのデータを読み書きする方法を説明します。

ファイルは大きく分けて、**テキストファイル**と**バイナリファイル**の2種類があります。

「テキストファイル」とは、テキストエディタで、データ内容の表示や編集、保存をすることができるファイルのことで、人にとって意味のある情報が書かれたファイルのことです。また、「バイナリファイル」とは、テキストエディタで、データ内容の表示や編集、保存ができないファイルのことで、コンピュータにとって意味のある情報が書かれたファイルのことです（詳しくは本文52ページ参照）。

例えば、ワープロで作成された文書ファイルは、バイナリファイルです。文書ファイルというと、いかにも人にとって意味のある情報が書かれているようですが、そうではありません。ワープロで作成された文書ファイルの中には、文書を装飾するための様々な情報が付加されています。これらの情報は、例えば、印刷された段階では人にとって意味のあるものになりますが、文書ファイルとしては、わけのわからない文字列の集合に過ぎないのです（テキストエディタで、ワープロの文書ファイルを開いてみてください）。

C言語に関わるテキストファイルといえば、次のものがあります。

ファイルの種類	拡張子	説明
C言語ソースファイル	.c	C言語のソースコードを記述したファイルのことです。
C言語ヘッダーファイル	.h	C言語の標準関数のプロトタイプ宣言を記述したファイルのことです。

また、バイナリファイルといえば、次のものがあります。

ファイルの種類	拡張子	説明
実行ファイル	.exe	C言語のソースコードをコンパイルして、コンピュータが実行できるようになったファイルのことです。

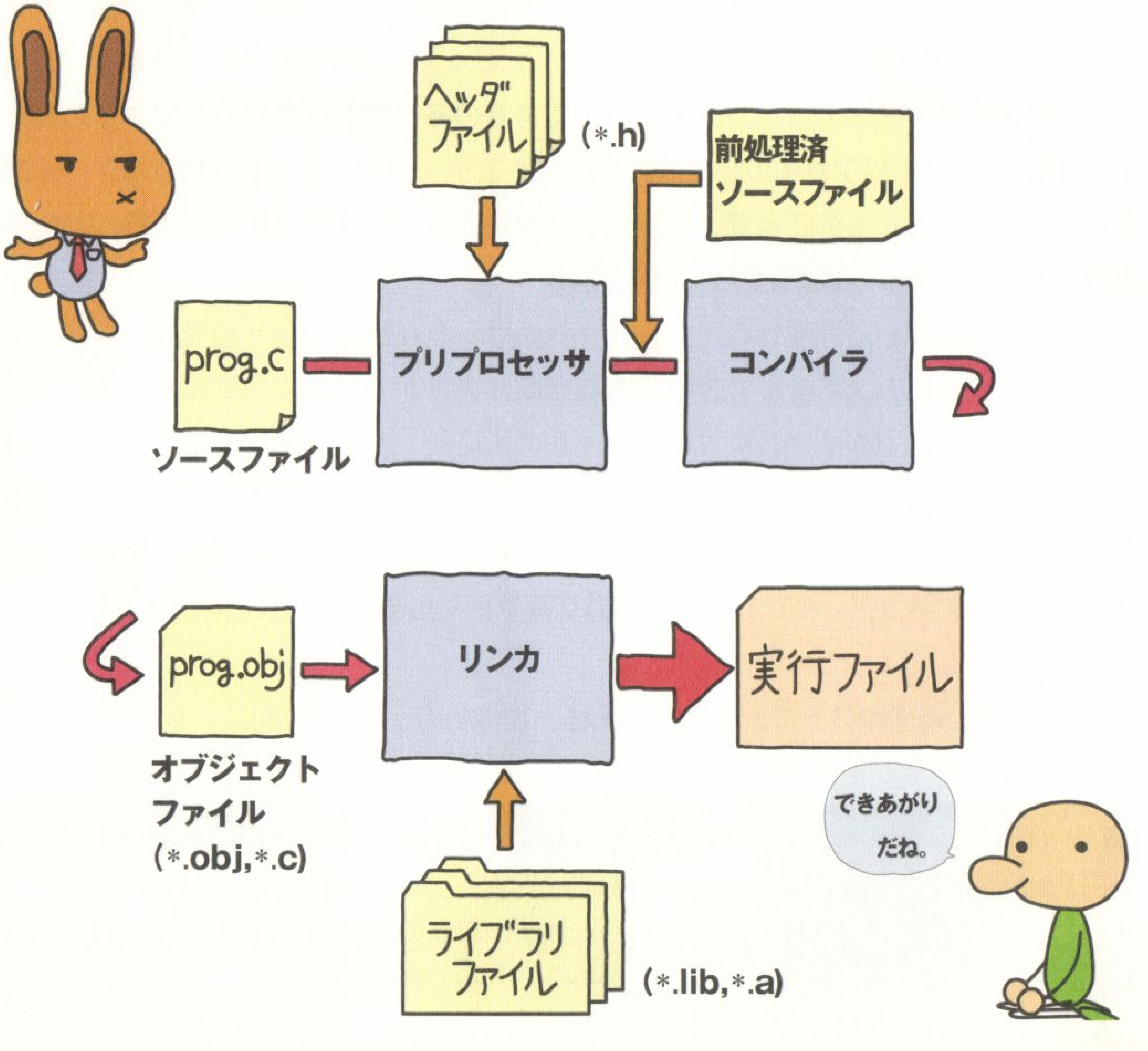
▶ **専門的な説明**

●実行ファイルができるまで

いままで詳しく触れていなかったのですが、ソースコードからどのようにして実行ファイルになっていくのかについて、ここで説明します。

C言語のソースファイルから、最終的に実行ファイルを作りたいわけですが、その流れをしっかりと覚えておくことは、これからプログラミングを続けていく人にとって、とても重要です。

C言語のソースコードから、実行ファイルができるまでの流れは、次の図のようになります。



図について、説明していきましょう。

作成したC言語のソースファイルは、**プリプロセッサ**で処理され、コンパイラにより**オブジェクトファイル**に変換され、続いて**リンカ**により、他のオブジェクトファイルやライブラリと結合されて、最後に実行ファイルが作られます。

なお、普通にC言語といった場合は、Cプリプリセッサとコンパイラ、リンカ、**ライブラリアン**（ライブラリの作成や修正を行うためのもの）のことを指します。

ファイルの種類	説明
プリプロセッサ	C言語のソースファイルに対して、コメントの削除やヘッダファイルのインクルードなどを行うものです。
前処理済みソースファイル	プリプロセッサにより処理されたファイルのことです。
コンパイラの処理内容	前処理済みのソースファイルをコンパイルして、オブジェクトファイルに変換します。この段階でソースコードは、機械語に翻訳されます。
オブジェクトファイル	基本的には、機械語のプログラムとなっているために、どのような言語から生成されたものであってもかまいません。オブジェクトファイルは、実行時に必要である「プログラムのどの部分を、メモリのどこに配置すればよいか」などの情報がないところが、実行ファイルと違うところです。
ライブラリファイル	オブジェクトファイルから、ライブラリアンを使って作成されます。オブジェクトファイルの場合は、そのすべてが実行ファイルに組み込まれるのに対して、ライブラリの場合は、必要な部分だけを取り出して使えるような形式になっています。C言語の標準関数のほとんどは、この形式で提供されています。
リンカ	複数のオブジェクトファイルを結合し、その中で利用しようとしている関数をライブラリから検索して必要なものだけを組み込みます。

ファイルの操作 ❶

ここからは、主にテキストファイルのデータファイルに関して、作成、書き込み、読み込みなどの方法を説明していきます。

ファイルの作成や読み書きをするためには、次のような手順が必要です。

▶作成手順

❶ ファイルポインタを宣言します。

❷ ファイルを開きます。

❸ ファイルにデータを書いたり、ファイルからデータを読んだりします。

❹ ファイルを閉じます。

この手順は、次のように考えればわかりやすいでしょう。

▶作成手順

❶ たくさんあるノートから、１つのノートを用意します。
❷ ノートを開きます。
❸ ノートに読み書きをします。
❹ ノートを閉じます。

それでは、この基本手順を踏まえて、ソースコードを書いていきます。はじめは、ファイルを開くことと、閉じることをおぼえましょう。

次のソースコードは、「ファイルを開くことの成功、不成功」と「ファイルを閉じることの成功、不成功」を表示します。

```
/* MyC211.c */

#include<stdio.h>

int main(void)
{
  FILE *fp;  ──→ファイルポインタを宣言します
  int a;

  fp = fopen("MyC.c" ,"r");  ──→ファイルを開きます

  if(fp == NULL){
         printf("I can't open a file.¥n");
  }else{
         printf("I can open a file.¥n");

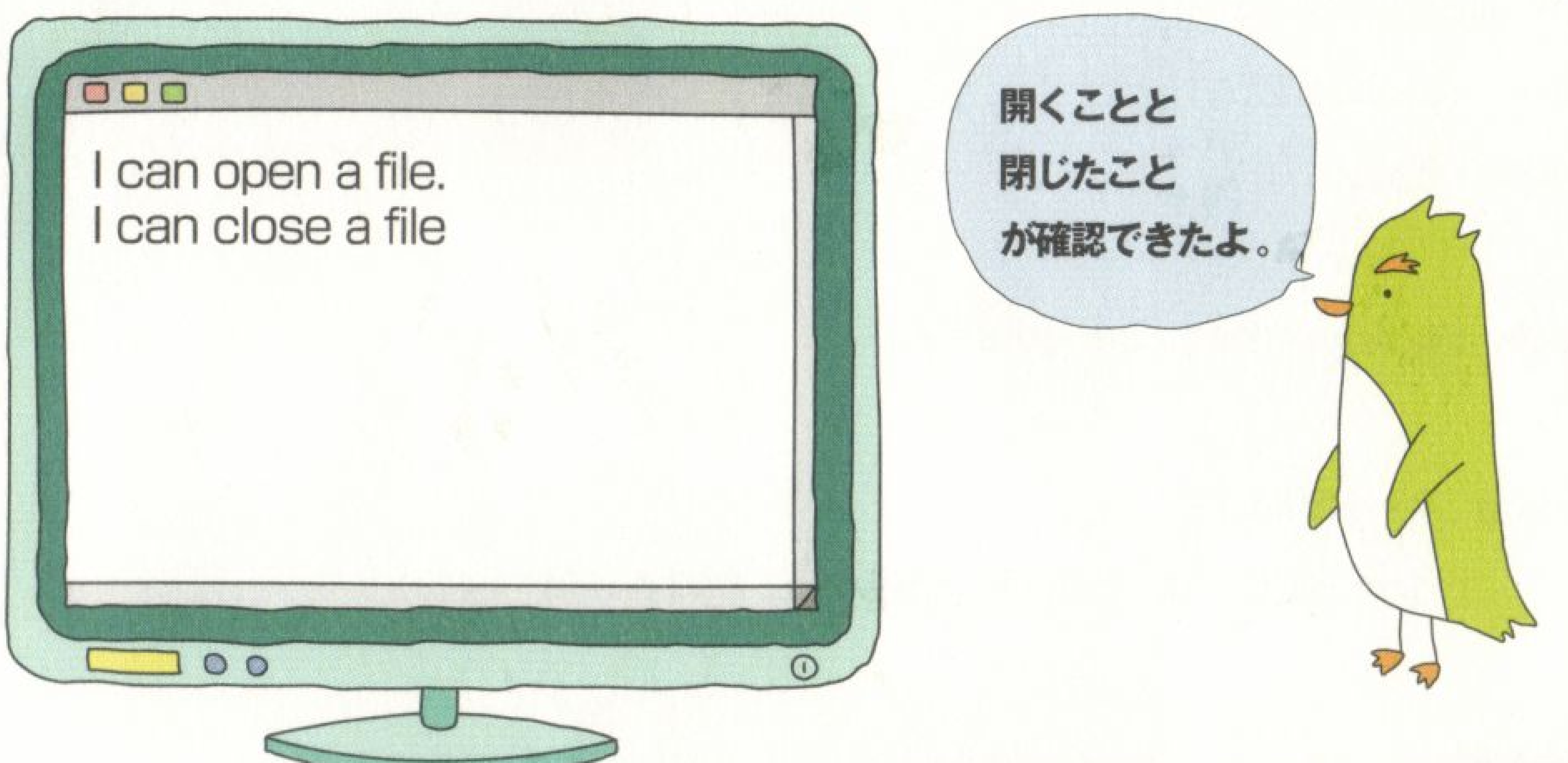
         a = fclose(fp);  ──→ファイルを閉じます

         if(a != 0){
                printf("I can't close a file.¥n");
```

```
21:          }else{
22:                  printf("I can close a file¥n");
23:          }
24:
25:  }
26:}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC211.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「`cd c:¥MyC` Enter」と入力してください。
❺「`gcc MyC211.c -o MyC211` Enter」と入力します。
❻「`MyC211` Enter」と入力します。



7行目で、ファイルポインタを宣言しています。

10行目で、「fopen」を使ってファイル名「MyC.c」というソースファイルを読み込みます。「fopen」の引数は、1番目がファイル名、2番目がファイルを開くためのモードです。ファイルを開くためのモードは数種類あり、今回は、読み込み(READ)を示す「r」を指定します。

このときファイルはすでに存在している必要があります。もし、存在しないファイルを「"r"」で開こうとすると、「fopen」オープン失敗を示すヌルを返します。成功すれば、指定ファイルのファイルポインタを取得して、そのファイルポインタを使ってファイル操作することになります。

「fopen」でファイルポインタの取得ができれば、ファイル操作をする場合、ファイル名

を使わずファイルポインタを使います。

17行目で、すべてのファイル操作が終了したら、「fclose」でファイルを閉じます。

ファイルの操作❷

前の項では、ファイルのオープンとクローズについて説明しました。ここでは、読み込んだファイルの内容を読み取ることについて説明します。

次のソースコードは、読み込んだファイルから文字列を読み取ります。

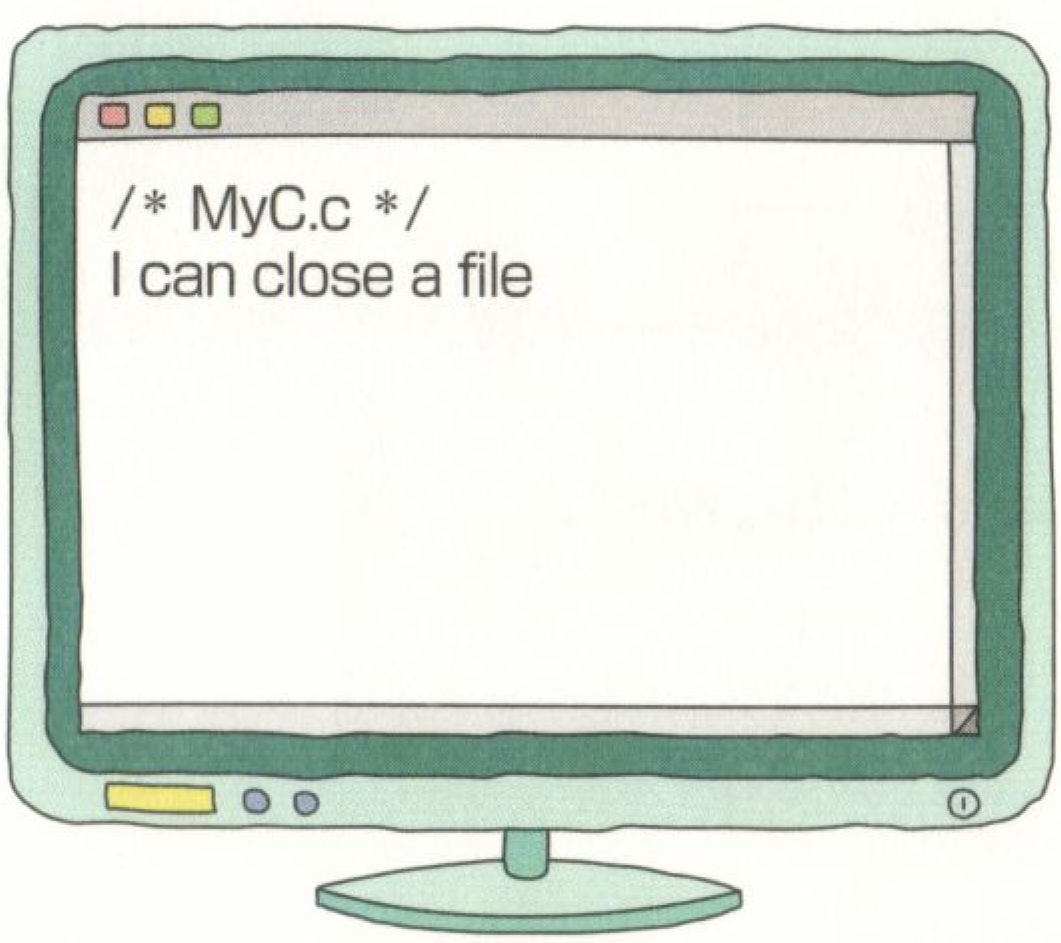
```
 1:/* MyC212.c */
 2:#include<stdio.h>
 3:
 4:int main(void)
 5:{
 6:  int a;
 7:  char  Scode[20];
 8:  FILE *fp;
 9:
10:  fp = fopen("MyC.c" ,"r");
11:
12:  if(fp == NULL){
13:          printf("I can't open a file.¥n");
14:
15:  }else{
16:
17:          fgets(Scode, 20, fp);
18:          printf("%s",Scode);
19:
20:          a = fclose(fp);
21:
22:          if(a != 0){
23:                  printf("I can't close a file.¥n");
```

文字列を読み取ってScodeに代入します

```
24:            }else{
25:                    printf("I can close a file.¥n");
26:            }
27:    }
28:}
```

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC212.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC212.c -o MyC212** Enter」と入力します。
❻「**MyC212** Enter」と入力します。





17行目で、「fgets」を使ってファイルから文字列を読み取り、「Scode」に代入しています。「fgets」の引数は、1番目が読み込んだ文字列の格納先、2番目がファイルから読み込む文字数、3番目が読み込みたいファイルのファイルポインタです。読み取りは、行末文字（'¥n'）に出会うか、n-1個の字を読み込むまで行われます。文字列の最後には「¥0」が付加され、通常は「¥n」も入力されます。

ファイルの操作❸

ここからは、ファイルへの書き込みについて説明していきます。
次のソースコードは、ファイルに文字列を書き込むものです。

```
#include<stdio.h>

int main(void)
{
    int a;
    char s[256];
    FILE *fp;

    printf("Input > ");
    scanf("%s",s);

    fp = fopen("sample.txt", "w");

    if(fp == NULL){
            printf("I can't open a file.¥n");

    }else{
            fputs(s, fp);

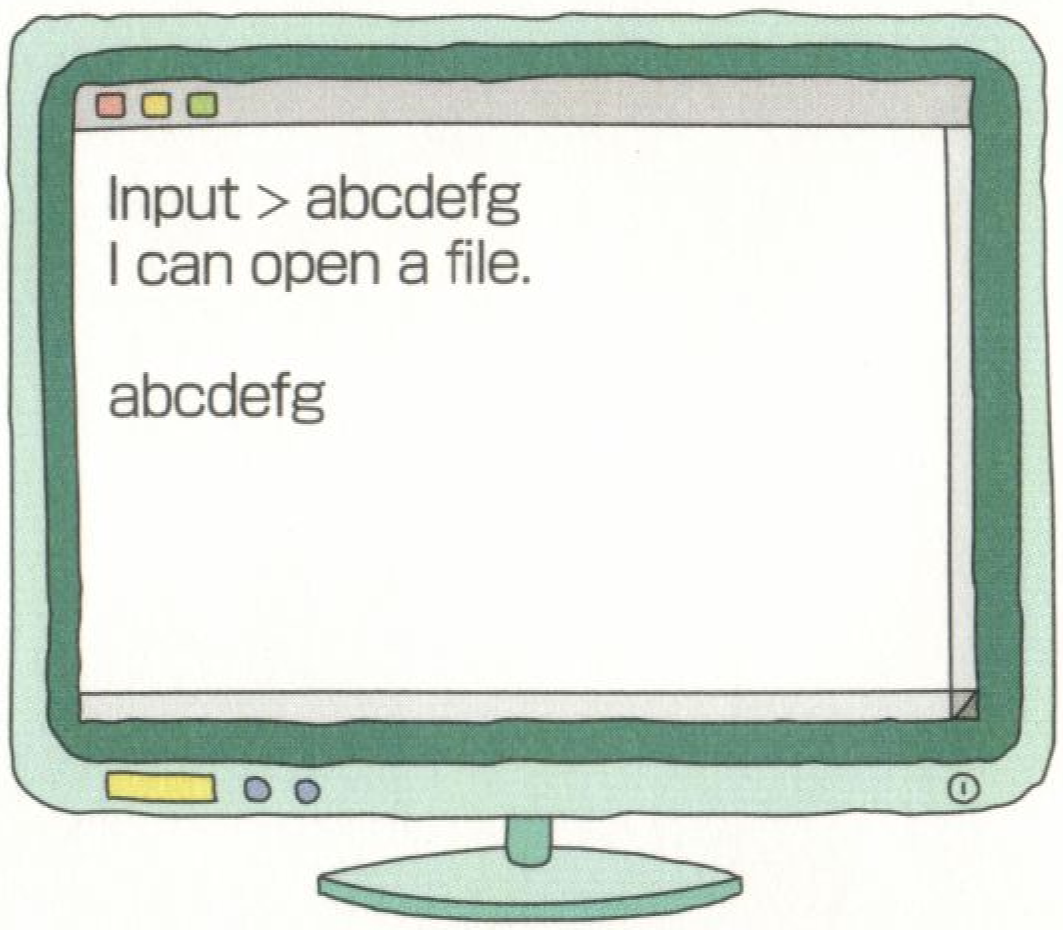
            a = fclose(fp);
            if(a != 0){
                    printf("I can't open a file.¥n");
            }else{
                    printf("I can open a file.¥n");
            }
    }
}
```

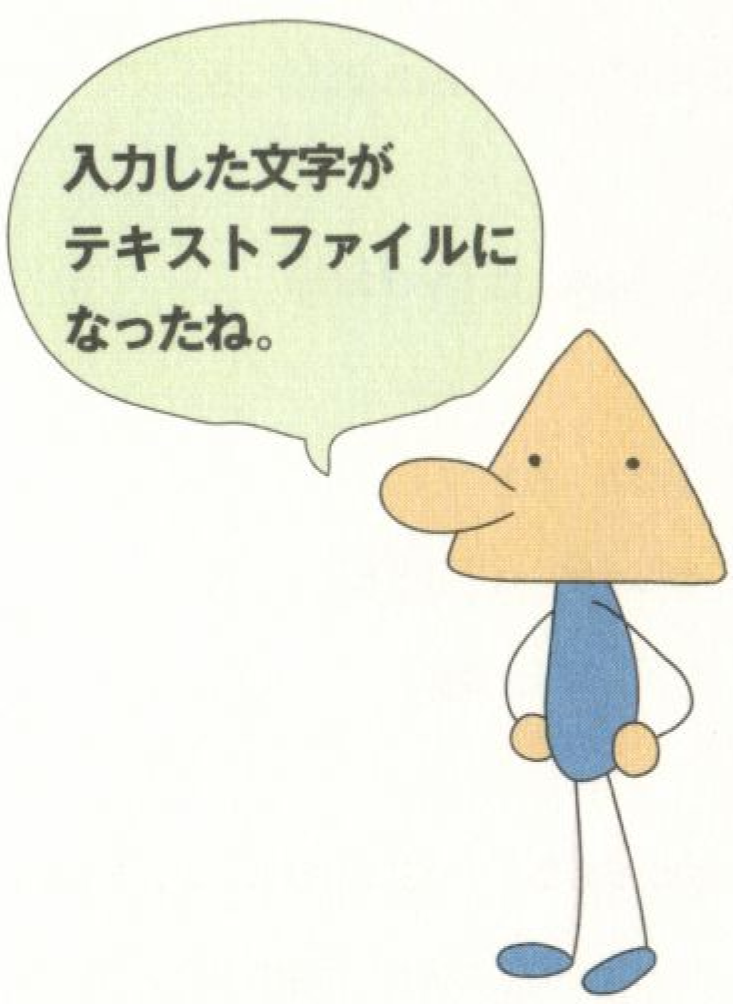
（12行目）ファイルを開くためのモードを書き込みモードにします

（18行目）ファイルに、文字列「s」の内容を書き込みます

▶作成手順と実行結果

❶ テキストエディタでソースコードを入力します。
❷「MyC」フォルダにファイル名「MyC213.c」で保存します。
❸ コマンドプロンプトを起動します。
❹「**cd c:¥MyC** Enter」と入力してください。
❺「**gcc MyC213.c -o MyC213** Enter」と入力します。
❻「**MyC213** Enter」と入力します。
❼「**abcdefg** Enter」と入力します。





12行目で、「fopen」を使って、ファイル名「sample.txt」というデータファイルを新規に作成して開いています。ファイルを開くためのモードは書き込みなので、書き込み(WRITE)を示す「w」としています。

18行目で、ファイルに文字列「s」の内容を書き込んでいます。「fputs」の引数は、1番目が書き込む文字列の格納先、2番目が書き込みたいファイルのファイルポインタです。「¥0」は書き出されません。

今回は、作成手順で「abcdefg Enter」と入力していますので、ファイル「sample.txt」の内容は「abcdefg」になっているはずですので、確認してみましょう。

確認の仕方は、コマンドプロンプトで次のようにします。

▶作成手順

❶「**cd c:¥MyC** Enter」と入力してください。
❷「**type sample.txt** Enter」と入力します。

「type ファイル名」は、ファイルの内容を画面に表示するコマンドです。

例題 ファイルの操作

ファイル名「sample2.txt」というファイルを新規に作成して、そのファイルに「0123456789」と書き込み、ファイルを閉じたあとに、再度ファイル「sample2.txt」を開き、そこからファイル内容を読み込むソースコードを書いてください。

```
#include<stdio.h>

int main(void)
{
  int a;
  char s[256];
  FILE *fp;

  printf("Input > ");
  scanf("%s",s);

  fp = fopen("sample2.txt", "w");
  if(fp == NULL){
         printf("I can't open a file.¥n");

  }else{
         fputs(s, fp);

         a = fclose(fp);
         if(a != 0){
                printf("I can't open a file.¥n");
         }else{
                printf("I can open a file.¥n");
         }
  }
```

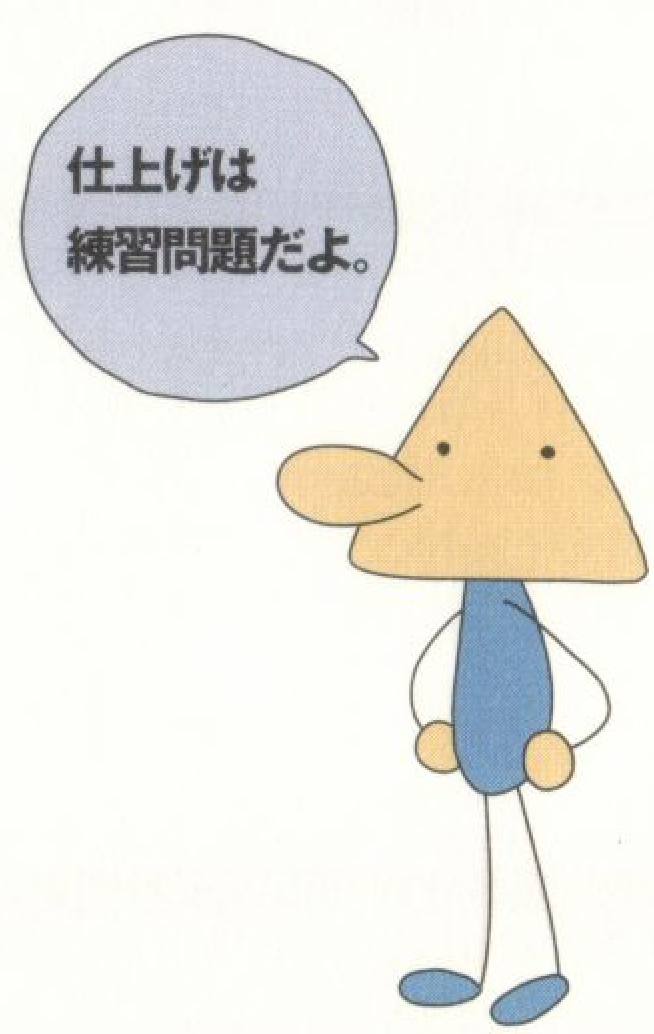
```
fp = fopen("sample2.txt" ,"r");
if(fp == NULL){
        printf("I can't open a file.¥n");

}else{

        fgets(s, 20, fp);
        printf("Output > %s¥n",s);

        a = fclose(fp);

        if(a != 0){
                printf("I can't close a file.¥n");
        }else{
                printf("I can close a file.¥n");
        }
}
}
```



practice 練習問題 解答は213ページ

1 標準入力 難易度★★★★★

ソースコード「Practice10.c」を、標準入力を使って改良してみましょう。

▶▶ヒント：本文175ページ参照

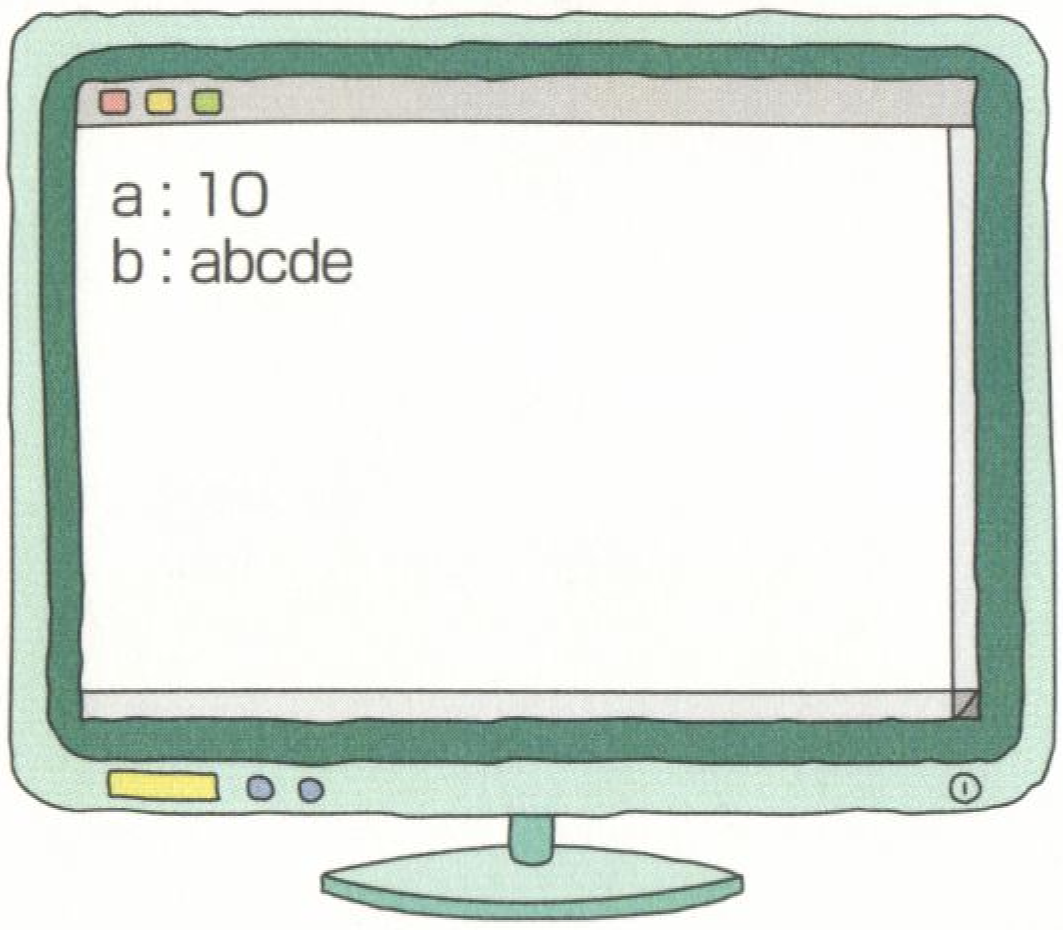
2 構造体 難易度★★★★★

次の構造体を使って、画面に以下のように表示するソースコードを書いてください。

▶▶ヒント：本文180ページ参照

▼構造体

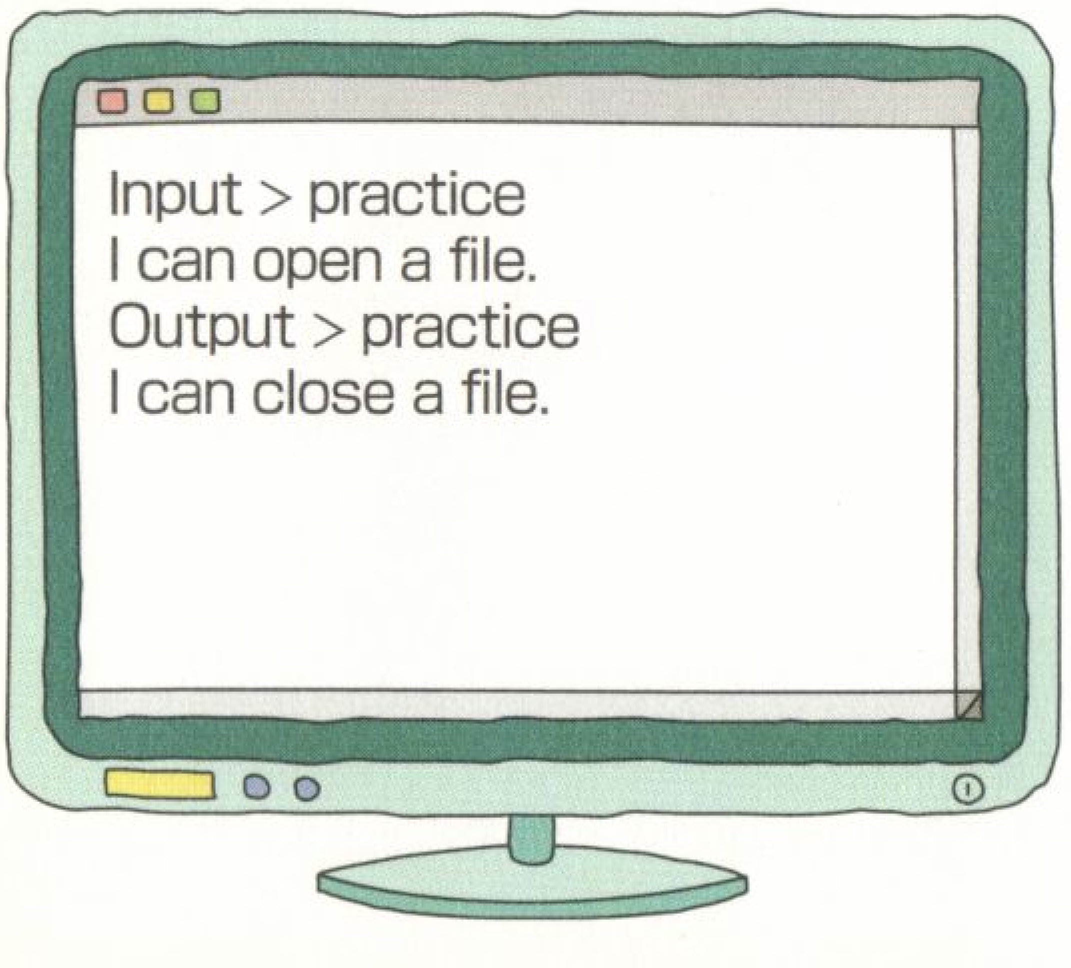
```
struct MyStruct {
        int a;
        char b[10];
};
```



3 ファイルの操作 難易度★★★★★

ファイル名「practice.txt」というファイルを新規に作成して、そのファイルに「practice」と書き込み、ファイルを閉じたあとに、再度ファイル「practice.txt」を開き、そこからファイル内容を読み込むソースコードを書いてください。画面の表示は、次のようになります。

▶▶ヒント：本文193ページ参照



Chapter7のまとめ

★文字

文字とは、1バイトの文字コードのことをいいます。「'A'」のように、シングルクォーテーションで囲みます。

★文字列

文字列とは、1文字以上の文字の集まりのことをいいます。「"ABC"」のように、ダブルクォーテーションで囲みます。

★標準入力

標準入力とは、キーボードでの入力のことをいいます。

★標準出力

標準出力とは、コマンドラインの画面での出力のことをいいます。

★標準入出力

標準入力とは、プログラムが標準に使う入出力のことをいいます。

★構造体

構造体とは、変数をグループ化したもののことをいいます。

★ファイル

ファイルとは、コンピュータによって処理されるデジタル情報としての、文字やデータ、プログラムなどのことをいいます。

★ファイル操作の手順

1. ファイルポインタを宣言します。
2. ファイルを開きます。
3. ファイルにデータを書いたり、ファイルからデータを読んだりします。
4. ファイルを閉じます。

★リンカ

リンカとは、リンクを行うプログラムのことをいいます。

★プリプロセッサ

プリプロセッサとは、コンパイラがソースコードをコンパイルする前に行う処理のことです。

★ライブラリ

ライブラリファイルとは、オブジェクトファイルから、ライブラリアンを使って作成されます。

COLUMN

構造体って？

```
struct MyStruct {        → 構造体の宣言
    char namae[10]
    int tosi;
    float sincho;
    float taijyu;
};
int main(void)
{
        ・・・

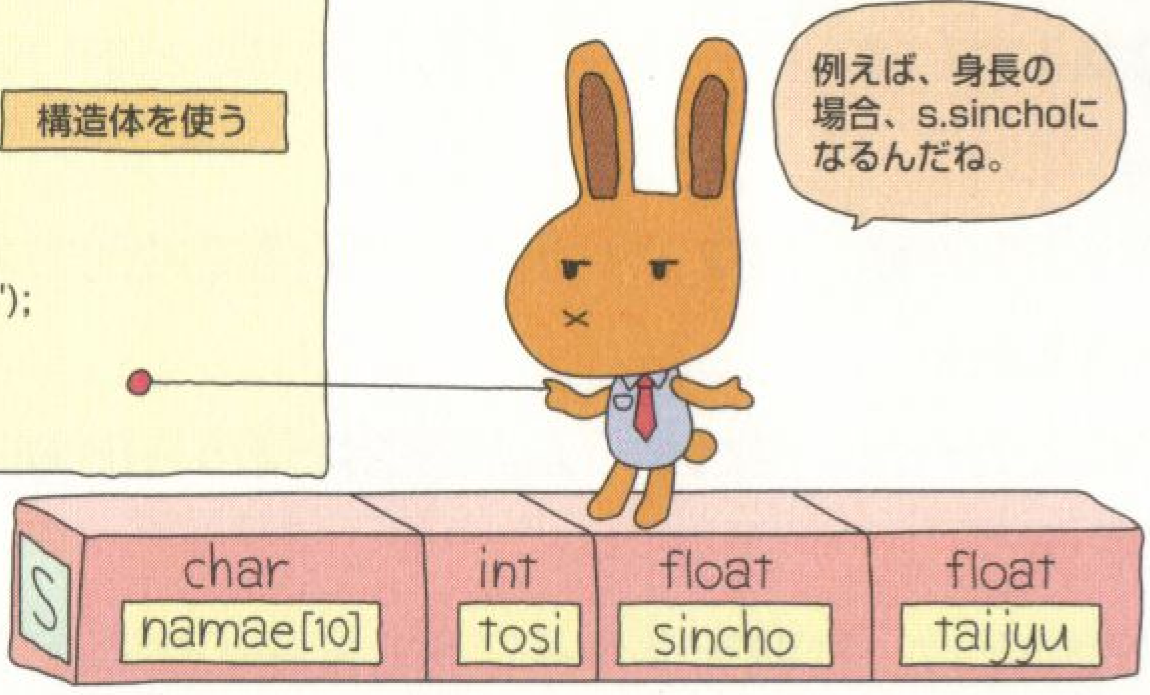
  struct MyStruct s;    → 構造体を使う

        ・・・

  strcpy(s.namae,"usagi");
  s.tosi = 10;
  s.sincho = 120.5;
  s.taijyu = 20.5;
```

変数を使うときの書式

構造体変数．メンバ名

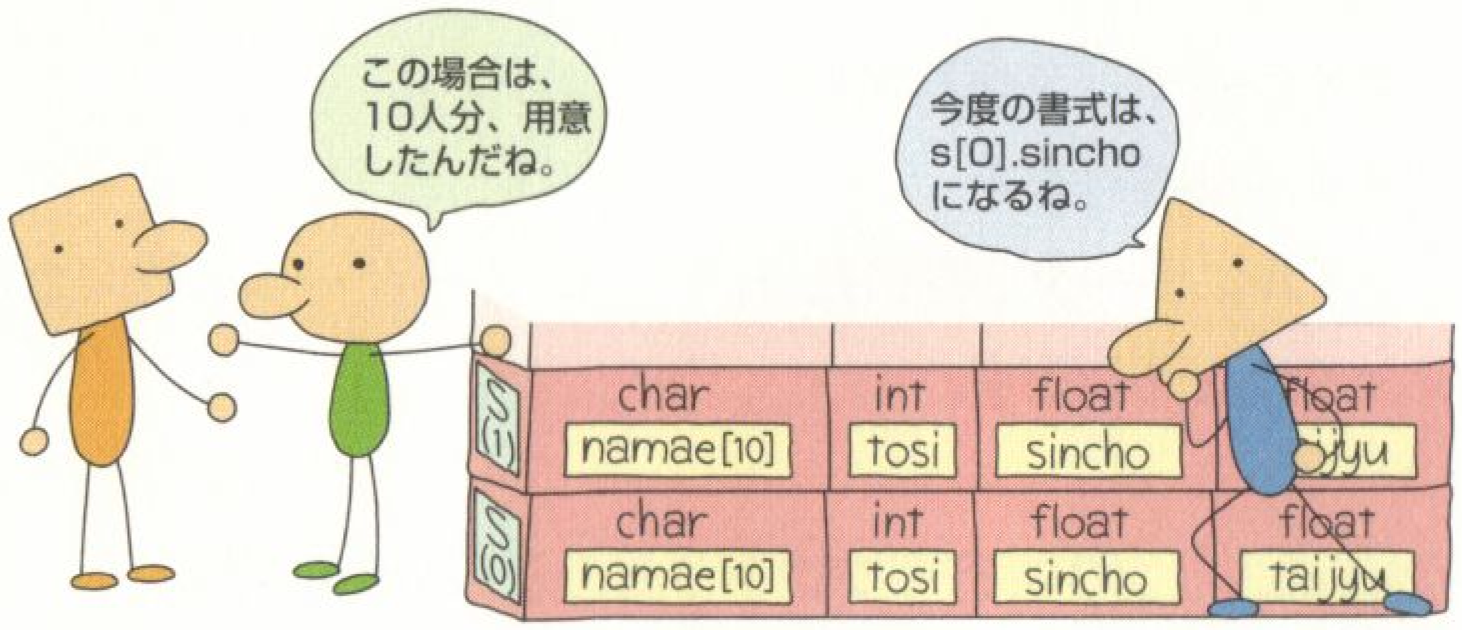


構造体配列を使うには

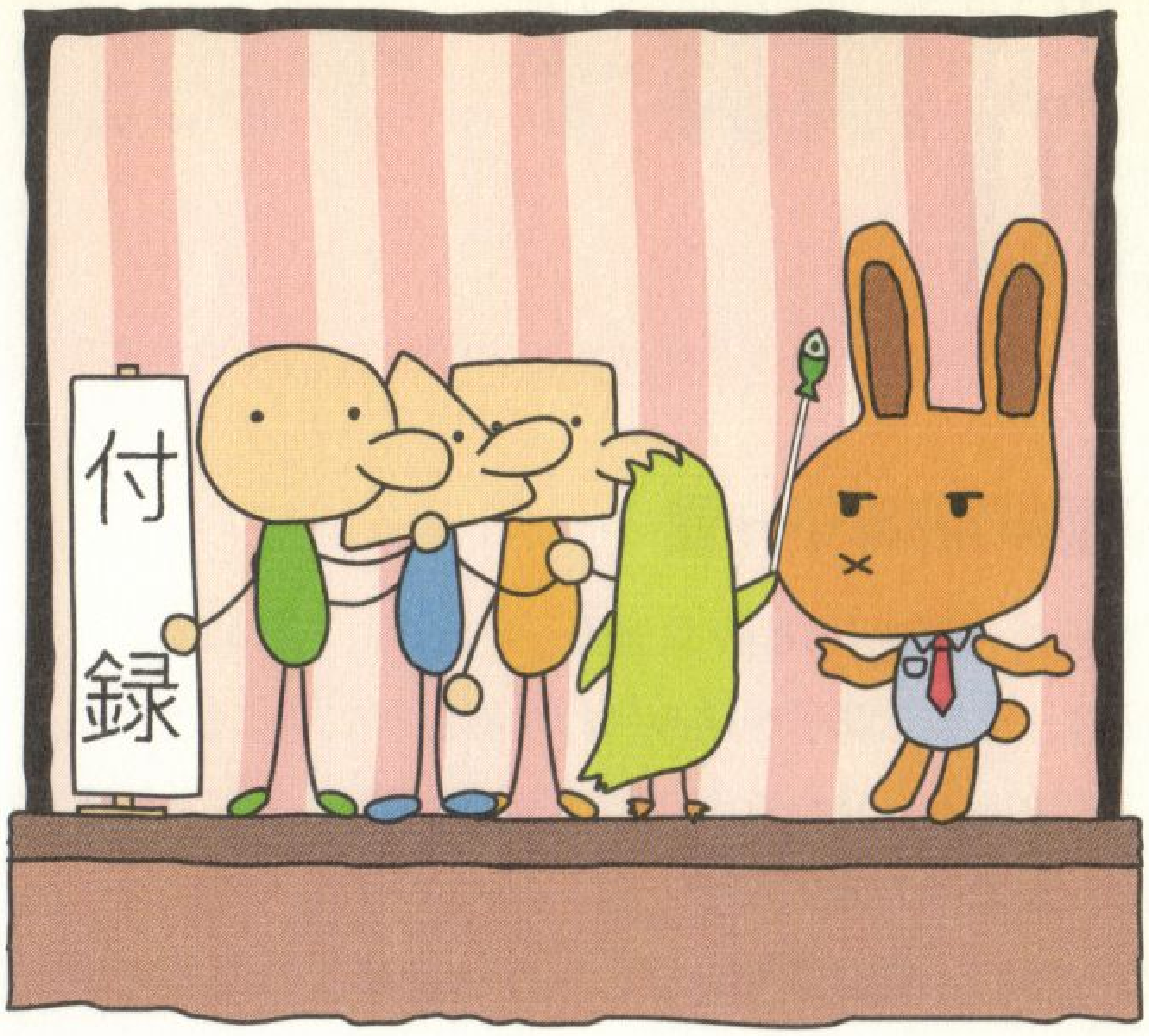
struct MyStruct s[10]; → 構造体配列を使う

変数を使うときの書式

構造体変数[i]．メンバ名



構造体は、いくつかの型の違う変数をひとまとめにして扱うための仕組みです。例えば、ある人の名前、年齢、身長や体重をまとめて扱いたいような場合に使います。



★付録には、「C言語の演算子、予約語一覧」、
「じゃんけんゲームサンプルプログラム」
があります。

付録3に練習問題の解答があります。

★予約語について復習したい人は

➡第3章本文58ページへ

★演算子について復習したい人は

➡第4章本文84ページへ

付録01 演算子、予約語一覧

C言語の演算子一覧

C言語は、数を扱う様々な演算子を用意しています。

計算は、この演算子を組み合わせることに他なりません。

ここでは、よく使われるC言語の演算子を、表によって詳しく説明していきます。

なお、演算子の**優先順位**とは、「+、−」よりも「*、/」、のほうが先といった演算子の優先度を示し、**結合規則**とは、右から演算するのか左から演算するのかといった「演算する/される」の関係のことです。また、単項演算子とは、自分自身に対して演算を行うものです（本文84ページ「▼算術演算子❷」で紹介した、「A++」や「A--」の「+、-」のことです）。

▼例）優先順位と結合規則

```
1 + 2 * 3
```

上記では、「*」の優先順位が、「+」よりも高いので、「2」と「3」を乗算（かけ算）したあとに、「1」との加算（足し算）を行います。

これを、「1」と「2」加算した後に、「3」と乗算するように変えるには、次のように書きます。

```
(1 + 2) * 3
```

▼C言語の演算子一覧

優先順位	演算子	扱うデータ型	結合規則	演算の内容
1	+	数値	右	単項演算子のプラス
1	-	数値	右	単項演算子のマイナス
2	*	数値	左	乗算
2	/	数値	左	除算
2	%	数値	左	剰余
3	+	数値	左	加算
3	-	数値	左	減算

優先順位	演算子	扱うデータ型	結合規則	演算の内容
5	<	数値	左	より小さい
5	<=	数値	左	以下
5	>	数値	左	より大きい
5	>=	数値	左	以上
6	==	基本データ型	左	等しい
6	!=	基本データ型	左	等しくない
13	=	左辺は変数	右	代入

実は、代入（「=」）も演算子でした。

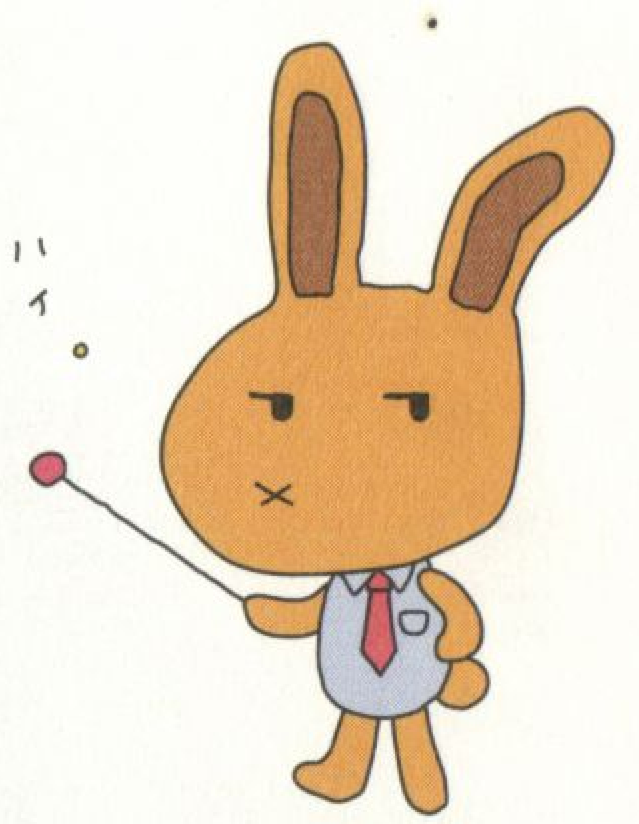
予約語一覧表（アルファベット順）

予約語とは、ソースコードを記述するときに基本となる、言語側が用意してくれている単語のことで、ユーザーがこの単語を勝手に使って（自分で作っては）はいけません。

▼C言語予約語一覧表（アルファベット順）

予約語	読み方	意味
auto	オート	自動変数（関数の中でのみ使うローカル変数）を明示的に宣言する場合に使う。
break	ブレイク	switch文や繰り返しを抜け出す。
case	ケイス	switchブロック内でラベル（「:」で終わる文字列）のようにして使用する。
char	キャラクタ	文字を表すための型、符号付8ビット整数。
const	コンスト	値が変化しないことを保証する。
continue	コンティニュー	繰り返し内で次の繰り返しに入る。
default	デフォルト	switchブロックの中でいずれのcaseにも該当しなかった場合の飛び先。
do	ドゥ	繰り返し条件があとにくる繰り返しの開始点を示す。
double	ダブル	倍精度浮動小数点を表すための型。
else	エルス	if文が成立しなかった場合に実行されることを示す。
enum	イナム	列挙型。
extern	エクスターン	グローバル変数や関数を呼び出す。
float	フロート	単精度浮動小数点を表すための型。
for	フォー	繰り返し。
if	イフ	条件分岐。

予約語	読み方	意味
int	イント	32ビット符号付き整数を表すための型。
long	ロング	64ビット符号付き整数を表わすための型。
register	レジスタ	レジスタ変数。
return	リターン	関数の終了。
short	ショート	16ビット符号付き整数を表すための型。
signed	サインド	明示的に符号を付ける。
sizeof	サイズオブ	データ型の大きさを示す。
static	スタティック	恒久変数領域を示す。
struct	ストラクト	構造体。
switch	スイッチ	他方分岐。
typedef	タイプデフ	定義されている型の指定を付け替える。
union	ユニオン	共用体
unsigned	アンサインド	符号なし。
void	ボイド	戻り値がないことを示す。
volatile	ボラタイル	コンパイラの最適化の制御。
while	ホワイル	ループの繰り返し条件。



付録02 じゃんけんゲームサンプルプログラム

C言語でじゃんけんゲームを作ろう

●概要

対戦するうちに、コンピュータが強くなるじゃんけんゲームです。

「じゃんけんゲームサンプルプログラム」は、次のWeb上でダウンロードすることができます。

URL:http://www.shuwasystem.co.jp/books/7980/0542-8/0542-8.html

●説明

じゃんけんを繰り返していくうちに、コンピュータがあなたのじゃんけんのパターンを覚えて、段々強くなっていく「じゃんけんプログラム」を作ります。

まずは、「1つ前のあなたの手」を表にして覚える方法を考えてみましょう。次のような表（配列）を作って、この表（配列）の該当するところの値を、毎回「+1」していきます。

		いまのあなたの手		
		ぐー[0]	ちょき[1]	ぱー[2]
1つ前のあなたの手	ぐー [0]	[0][0]	[0][1]	[0][2]
	ちょき[1]	[1][0]	[1][1]	[1][2]
	ぱー [2]	[2][0]	[2][1]	[2][2]

例えば、「1つ前のあなたの手」が「ぐー」で、「いまのあなたの手」が「ちょき」だったら、「[0][1]」の値に「+1」します。さらに、じゃんけんをするたびに、該当する表（配列）の値に「+1」していきます。

その結果、あなたは「ちょき」のあとに「ぐー」を出しやすいとか、「ぐー」のあとに「ちょき」を出しやすいとかが、表（配列）のそれぞれの値を比べることで、予測できるようになりますね。

例えば、「ぐー」のあとの手を予測するには、「1つ前のあなたの手」の「ぐー」欄で「いまのあなたの手」のそれぞれの値を比べれば、あなたが次に出しやすい手を予測できるわけです。

また、勝ち負けの判定については、「ぐー」、「ちょき」、「ぱー」を、それぞれ、「0」、「1」、「2」と考えて、あなたとコンピュータの勝ち負けを、表にすると次のようになります。

		あなた		
		ぐー (0)	ちょき (1)	ぱー (2)
コンピューター	ぐー (0)	あいこ	勝ち	負け
	ちょき(1)	負け	あいこ	勝ち
	ぱー (2)	勝ち	負け	あいこ

この表の「勝ち負け」は、「コンピュータの手 - あなたの手 + 3」を3で割ったときの余りで、表すことができます。つまり、余りが「0」のときは「あいこ」、「1」のときは「あなたの負け」、「2」のときは「あなたの勝ち」となりますね。

```
/* Janken.c */
#include <stdio.h>

int main(void)
{
        int hmn; /* あなたの手 */
        int hmb; /* あなたの前の手 */
        int cp;  /* コンピュータの手 */
        int jg;  /* 審判 */
        int tTable[3][3]; /* 戦略表 */
        int i;
        int j;

        hmb = 0; /* はじめは、前の手がない */
        for(i = 0; i < 3; i++){
                for(j = 0; j < 3; j++){
                        tTable[i][j] = 0;
                }
        }

        do{
                printf("1:Gu 2:Choki 3:Pa 0:End¥n");
                scanf("%d", &hmn);
```

```
        if (hmn == 0) { /* 入力されたのが「0」なら終了*/
            break;
        }
        hmn = hmn - 1; /* 配列用に減算 */

        switch (hmn) {
            case 0:
                printf("You      :  Gu¥n");
                break;
            case 1:
                printf("You      :Choki¥n");
                break;
            case 2:
                printf("You      :  Pa¥n");
        }

        if (tTable[hmb][0]>tTable[hmb]➡
        ➡[1] && tTable[hmb][0]> tTable[hmb]➡
            ➡[2]) { /* あなたが、ぐーを出しやすい */
            cp = 2;
            printf("Computer:  Pa¥n");
        }else if (tTable[hmb][1]>tTable[hmb]➡
         ➡[2]){ /* あなたが、ちょきを出しやすい */
            cp = 0;
            printf("Computer:  Gu¥n");
        }else { /* あなたが、ぱーを出しやすい */
            cp = 1;
            printf("Computer:Choki¥n");
        }

        jg = (cp - hmn + 3) % 3; /* 審判 */
        switch(jg){
```

※注意　ソースコード内の➡は、レイアウトの都合により行を折り返していますが、実際には1行で入力してください。

```
                    case 0:
                            printf("Aiko¥n");
                            break;
                    case 1:
                            printf("Kachi¥n");
                            break;
                    case 2:
                            printf("Make¥n");
            }

            tTable[hmb][hmn] = tTable[hmb][hmn]➡
                                    ➡+ 1; /* 戦略表の更新 */
            hmb = hmn;

    }while( (hmn == 0) || (hmn == 1) ➡
                              ➡|| (hmn == 2) );
}
```



※注意　ソースコード内の➡は、レイアウトの都合により行を折り返していますが、実際には1行で入力してください。

付録03 練習問題の解答

第1章練習問題

1 Java、C++、BASIC、Pascal、Fortranなど

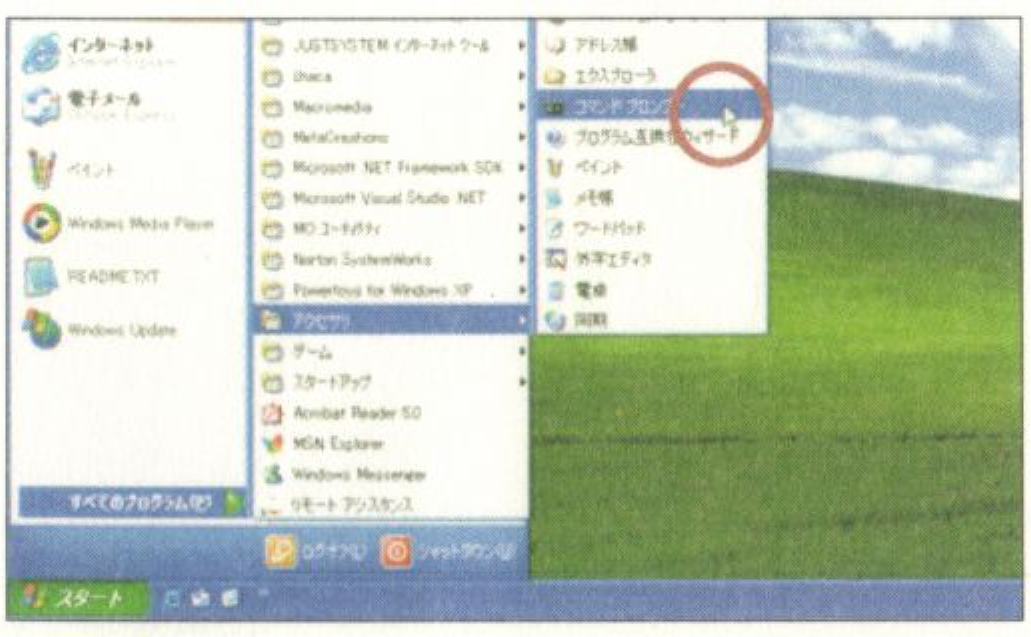
2 機械語、アセンブラ言語、高級言語

3 インタプリタやコンパイラの役目は、人間のわかりやすいプログラム（ソースコード）から、機械がわかりやすいプログラム（命令）に変換することです。
なお、この二つの違いは、1行1行変換して実行するか（インタプリタ）、一括して変換するか（コンパイラ）です。

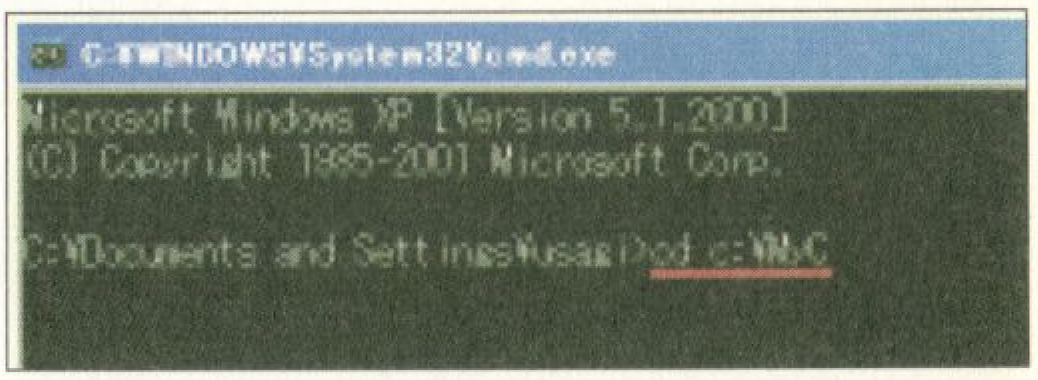
第2章練習問題

1 テキストエディタと違ってワープロは、文字の書式情報も扱います。

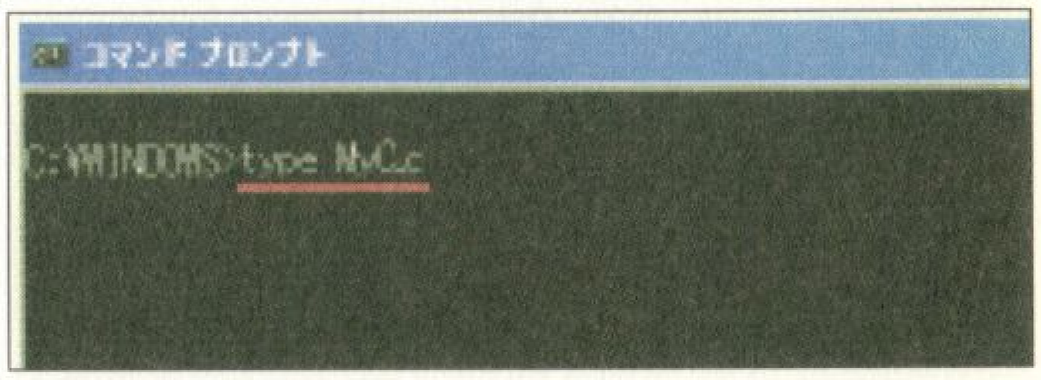
2 ❶**コマンドプロンプト**を起動します。
スタートメニュー➡**プログラム**➡**アクセサリ**➡**コマンドプロンプト**をクリックしてください。



❷**コマンドプロンプト**に、
「**cd C:¥MyC** [Enter]」と入力します。



❸**コマンドプロンプト**に、「**type MyC.c** [Enter]」と入力すると、ファイル内容が表示されます。



3 ❶ × ❷ × ❸ ○

第3章練習問題

1 ❶テキストエディタ(「メモ帳」)を起動します。
❷**ファイル**メニュー ➡**開く**で、プログラミング用フォルダ「MyC」から「MyC .c」というファイルを開きます。
❸ソースコードの「MyC」とある部分を、「MyStudy」に直します。
❹**ファイル**メニュー➡**名前をつけて保存**をクリックして、プログラミング用フォルダ「MyC」に、「MyStudy.c」というファイル名で保存します。
　保存したら、メモ帳を終了させましょう。
❺コマンドプロンプトを起動します。
❻「**cd c:¥MyC** [Enter]」と入力してください。
❼「**gccMyStudy.c -o MyStudy** [Enter]」と入力します。

❽「MyStudy Enter」と入力します。

2 この練習問題には、解答は付けません。例と同じように、手順に分けて書いてみてください。

3 C言語のソースコードには、必ず、この「main関数」が必要で、C言語のプログラムは、このmainメソッドから処理が始まるようになっています。

第4章練習問題

1 ▼値の入れ替えのプログラム

```
/* Practice9.c */
#include <stdio.h>

int main(void)
{
        int a;
        int b;
        int c;

        a = 2;  ──→ aに値を代入
        b = 5;  ──→ bに値を代入
        printf("a = %d¥n", a);
        printf("b = %d¥n", b);
        printf("値の入れ替え");
        c = a;  ──→ cに値を代入
        a = b;  ──→ aにbの値を代入
        b = c;  ──→ bにcの値を代入
        printf("a = %d¥n", a);
        printf("b = %d¥n", b);

        return 0;
}
```

2 ▼円周と面積を求めるプログラム

```
/* Practice1.c */
#include <stdio.h>

int main(void)
{
        double r;
        double a;
        double b;
```

```
        double pi;

        r = 10.0;
        pi = 3.141592654;  ——→ 円周率

        a = 2.0 * pi * r;
        b = pi * r * r;

        printf("r = %f¥n", r);
        printf("Circumference = %f¥n", a);
        printf("Area = %f¥n", b);

        return 0;
}
```

3 ▼分を時間と分で表示するプログラム

```
/* Practice4.c */
#include <stdio.h>

int main(void)
{
        int a;

        a = 90;

        printf("%dmin = %dh%dm¥n", a, (a¥60), (a%60));

        return 0;
}
```

第5章練習問題

1 ▼条件分岐のプログラム

```
/* Practice3.c */
#include <stdio.h>

int main(void)
{
        int a;
```

```
        a = 12;

        if (a >= 7 && a <= 12){  → 7歳以上12歳以下
                printf("elementary school¥n");
        }else if(a >= 13 && a <= 15){  → 13歳以上15歳以下
                printf("junior high school¥n");
        }else{  → その他
                printf("unknown¥n");
        }

        return 0;
}
```

2 ▼九九のプログラム

```
/* Practice5.c */
#include <stdio.h>

int main(void)
{
        int i;
        int j;

        for (i=1; i<=9; i++) {  → かける数
                for (j=1; j<=9; j++){  → かけられる数
                        printf("%d x %d = %d¥n", i, j, (i*j));
                }
        }

        return 0;
}
```

3 ▼繰り返しと配列のプログラム

```
/* Practice6.c */
#include <stdio.h>

int main(void)
{
        int a[6];
        int i;
```

```
        for (i = 0; i<=5; i++) {
                a[i] = i * 20;
                printf("a[%d] = %d¥n", i, a[i]);
        }

        return 0;
}
```

第6章練習問題

1 ▼関数を作るプログラム

```
/* Practice7.c */
#include <stdio.h>

void mtoh(int a) ——→ 関数名 mtohは「分から時間へ」を表す
{
        printf("%dmin = %dh%dm¥n", a, (a¥60), (a%60));
}

int main(void)
{
        int a;

        a = 90;
        mtoh(a);

        return 0;
}
```

2 ▼関数を作るプログラム

```
/* Practice8.c */
#include <stdio.h>

void school(int a)
{
        if (a >= 7 && a <= 12){
                printf("elementary school¥n");
        }else if(a >= 13 && a <= 15){
                printf("junior high school¥n");
```

```
	}else{
		printf("unknown¥n");
	}
}

int main(void)
{
	int a;

	a = 12;
	school(a);

	return 0;
}
```

3 できない。
理由：
「int *p」では、ポインタが、int型なのではなく、int型の変数を指し示すために使うポインタ「p」を宣言したということです。つまり、どこかにあるint型の変数のアドレスしか、代入できません。

第7章練習問題

1 解答例

▼標準入力と条件分岐のプログラム

```
/* Practice10.c */
#include <stdio.h>

int main(void)
{
	int a;

	scanf("%d", &a);
	printf("a = %d¥n", a);

	switch (a) {
		case 0:
			printf("x¥n");
```

```
            break;
        case 1:
            printf("o¥n");
            break;
        default:
            printf("?¥n");
    }
}
```

2 解答例

▼構造体のプログラム

```
/* Practice11.c */
#include <stdio.h>
#include <string.h>

struct MyStruct {
    int a;
    char b[10];
};

int main(void)
{
    int i;
    struct MyStruct s;

    s.a = 10;
    strcpy(s.b, "abcde");

    printf("a : %d¥n", s.a);
    printf("b : %s¥n", s.b);

    return 0;
}
```

3 解答例

▼ファイル入出力のプログラム

```
/* Practice12.c */
#include<stdio.h>
```

```
int main(void)
{
	int a;
	char s[256];
	FILE *fp;

	printf("Input > ");
	scanf("%s",s);

	fp = fopen("practice.txt", "w");
	if(fp == NULL){
		printf("I can't open a file.¥n");

	}else{
		fputs(s, fp);

		a = fclose(fp);
		if(a != 0){
			printf("I can't open a file.¥n");
		}else{
			printf("I can open a file.¥n");
		}
	}

	fp = fopen("practice.txt" ,"r");
	if(fp == NULL){
		printf("I can't open a file.¥n");

	}else{


		fgets(s, 20, fp);
		printf("Output > %s¥n",s);

		a = fclose(fp);

		if(a != 0){
			printf("I can't close a file.¥n");
		}else{
			printf("I can close a file.¥n");
		}
	}
}
```

→ ファイルを開くためのモードを書き込みモードにする（fopen("practice.txt", "w")）

→ ファイルを開くためのモードを読み込みモードにする（fopen("practice.txt" ,"r")）

目的別索引

INDEX1

1章 プログラミング言語C言語について

2章 プログラミングをはじめる前に

3章 プログラミングの基本

4章 プログラミングの心得❶

5章 プログラミングの心得❷

6章 関数とポインタ

7章 C言語を使う

用語別索引

INDEX2

あ行

か行

さ行

た行

な行

は行

ま行

や行 ら行 わ行

アルファベット

数字・記号

世界でいちばん簡単な
C(シー)言語のe(イー)本
C言語の基本と考え方がわかる本

2003年 6月 8日 第1版第1刷発行
2006年11月 5日 第1版第5刷発行

著 者 堀江(ほりえ) 幸生(さちお)

発行者 斉藤 和邦
発 行 株式会社秀和システム
〒107-0062
東京都港区南青山1-26-1
寿光ビル5F

イラスト 西川 美春
制 作 アンカー・プロ
フィルム出力 株式会社明昌堂

印刷所 三松堂印刷所株式会社

ISBN4-7980-0542-8 C3055

定価はカバーに表示してあります。
乱丁本・落丁本はお取りかえいたします。
本書に関するご質問については、ご質問の内容と住所、氏名、電話番号を明記のうえ、当社編集部宛FAXまたは書面にてお送りください。お電話によるご質問は受け付けておりませんのであらかじめご了承ください。
Printed in Japan

世界でいちばん簡単な
C言語のe（イー）本
C言語の基本と考え方がわかる本

ISBN4-7980-0542-8

C3055 ¥1300E

9784798005423

1923055013001

定価 （本体 1300円＋税）

世界でいちばん簡単なC言語のe（イー）本

堀江幸生＝著 Sachio Horie
with Anchor Pro

Contents



秀和システム